

رِسَالَةُ البَرْمَجَةِ بِإِبْدَاع

م. وائل حسن محمدعلي (أبو إياس)

بِسْمِ الله، و الصَّلاةُ و السَّلامُ عَلَي رَسُولِ الله، و عَلَي آلِهِ و صَحْبِهِ و مَنْ وَالاه،

رِسَالَةُ البَرْمَجَةِ بِإِبْدَاع

الإصدارة 1.1

كَتَبَها:

م. وائل حسن محمدعلي (أبو إياس)

إلى إمام الدنيا:

عَلِي بن حَزم الظَّاهِرِي الأَنْدلسي

غفر الله له ورفع درجاته في الجنة

و الشيخ العلَّامة:

أبي إسحاق الحُوَيْني الأَثَرِي

بارك الله في عمره

شكرٌ خاص

هذا الشكر الخاص أُوَجِّهُه (بعد الله عز و جل) إلي برنامج (نجوم العلـوم stars of science) و القـائمين عليـه مـن: مُمَـوِّلين، و مُنتِجيـن، و فريـق عمـلٍ و مُحَكِّمين، و كذلك رفاقي في المو سم الرابع؛ ذلك لأنه أجبرني علي تغييـر أ سـلوب عملي في المشروع في فترةٍ كنتُ أحْوَج ما أكون فيها لذلك التغيير، كما أنه أتاح لـي فر صـة مـن الصـعب أن تتكـرَّر مـن التفـر غ التـام، بعيـداً عـن المشـاغل اليوميـة و المُنغِّصـات الحياتيـة و تكـاليف التوا صـل الاجتمـاعي، بمـا كَفَـل لـي تفريـغ طـاقتي الذهنية كاملةً للمشروع، و هـو مـا أ دي بـه إلـي أن يقفـز قفـزاتٍ غيـر مُتوقَّعـةٍ بالنسبة للفترة القصيرة التي شاركتُ بها في المُسابَقة. و تَكَفَّل الرفاق و طـاقم العمـل بـالجو الحميم الذي جعل العمل أكثر مُتعة و سلاسة.

فشكراً لهم جميعاً

رخصة مُحتوَيات هذا الكتاب

الفهرست

القِسْمُ الأوّل:

التّوْطِئة

عن الواقع العِلمى الإسلامى

ليس هناك عصرٌ كانت الأمة الإسلامية فيه فى أضعف حالاتها و أشدها تدهوراً من هذا العصر؛ فالواقع أن الأمة الإسلامية اليوم هى أشد أمم الأرض ضعفاً و خنوعاً، و يتجلّى هذا فى كل مناحى الحياة الإسلامية بلا استثناءٍ واحد، بدءاً من التشرذم بين أقطار الإسلام و انفراد أهل كل قُطْرٍ بقُطْرِهم، و تفشّى الدعوات الشُّعوبية و القُطْرِية بين أفراد الأمة لتُردَم الهوية الإسلامية تحت رُكامٍ من صراخات أهل تلك الدعاوى البغيضة.

إلى انتشار كل خُلقٍ كريهٍ بين عامة المسلمين، من سلبيةٍ و خُنوعٍ إلى التباغض و التنافر حتى بين أفراد العائلة الواحدة (بل و خصوصاً بين أفراد العائلة الواحدة !)، إضافةً إلى ذلك الفساد الذى انتشر كالسوس لينخر فى بلاد الإسلام على مرأىً و مَسمعٍ من كل مسئول، بل و بمُشاركة و مُباركة كثيرٍ من هؤلاء المسئولين، الأمر الذي كشفته ثورات الربيع العربي المُبارَكة و ستكشف الأيام التالية أغلب ما خَفِي منه اليوم بإذن الله عز و جل.

فرأينا بأم أعيننا مدي فداحة الخسارة التي مُنينا بها من السرقات و الاختلاسات التي ولغ فيها من لم يُراعوا عهداً و لا أمانة، فسرقوا و نهبوا ما يكفي لجعلهم يعيشون هم و أحفاد أحفاد أحفاد أحفاد أحفادهم ملوكاً مُتوّجين ! و كأنه قد خُيِّل إليهم أن نقودهم ستصحبهم إلي الآخرة فأرادوا الاستزادة بلا شبع !

و لا تفسير لكل ذلك النَّهم للسرقة إلا القول المنسوب لرسول الله صلوات الله و سلامه عليه: (**مَنْهُومَـانِ لا يَشْبَعَانِ : مَنْهُومٌ في عِلْمٍ لا يَشْبَع, و مَنْهُومٌ في دُنْيَا لا يَشْبَع**)[1]

و ليس هذا بآخر السوء: فالزيادة فى الواقع العلمى المُخجِل للأمة؛ فاليوم يعيش المُسلِمون عالةً على غيرهم فى العلوم و المُنتَجات العلمية بكل المقاييس:

فلا نُنتِج بل نستهلك،

و لا نختار أجود المُستهلَكات بل نختار الأسرع ربحاً و الأفضل مكسباً،

و حينما نستهلك ما نستهلكه لا نُسمّيه باسمه الأصلى بل باسم السلعة التى استهلكناها، فيُصبِح اسم كل نظام تشغيلٍ: windows، و تُصبِح كل لعبةٍ رقميةٍ: atari، و مثل هذا كثير.

بل و الأنكى و الأشد إثارة للغيظ أننا نُنفِق الكثير و الكثير على توافه الأمور، و لا نُنفِق على العلوم إلا أقل القليل !، و حتي أقل القليل هذا لا نُنفِقه على العلم الحقيقى، بل على العلم من عينة "تعلم كيف تصنع أسطوانة ويندوز عليها صورك و برامجك"، و ما تبقى مما يُنفَق يتم إنفاقه على مرتبات موظفى الحكومة من النوعية التي صارت لا تعبأ لا بعلمٍ و لا بتقدمٍ و لا بأي شيءٍ يَمُتُّ لـ"وجع الرأس"هذا بِصِلة !

أما العلم الحقيقى فليس له مُعِينٌ من المسئولين و أهل الحَل و العَقْد، و ليس أمام المُبدِع الحقيقى إلا أن يبحث عَمَّن هم على شاكلته ليُشكِّلوا مع بعضهم البعض ثُلةً من المنبوذين و يُحاولوا تجاوز واقعهم المريض، أو أن يرضى بالواقع و يحاول جاهداً التكيف معه، و دَفْع نَفْسه دفعاً للإحساس أن كل الأمور علي ما يُرام، و أنه "ليس في الإمكان أبدع مما كان"، و متي نَجح في هذا عاش راضياً عن نفسه و عن الكون بما فيه بذات الطريقة التي يَرضي بها الدجاج عن عالمه.

و لم يكن مُستغرَباً و الحال هكذا أن نرى أن العالم الإسلامى بأسْرِه (و هو المُكوَّن مما يُقارِب الإثنين و خمسين دولةً دينها الأول الإسلام) يشهد انحطاطاً مُفزِعاً في جامعاته و معاهده و مدارسه و مواقعه التعليمية كلها تقريباً (إلا من رَحِم الله تعالي)، و أن يصل حال الإنتاج العلمي فيه في شتي المجالات إلي الحضيض.

و بينما يهتم أهل التربية و التعليم عندنا بتلقين الطلاب أكبر كميةٍ مُمكِنةٍ من المعلومات التي ليس لها أي أهميةٍ لا في حياتهم العادية و لا في مستقبلهم العلمي، يقوم العالَمُ المُتقدِّم بتحبيب طلابه في العلم و التحصيل العلمي و الإستمتاع بهما، و بينما يستمتع الطالب الغربي (أو علي الأقل لا يتم تعذيبه بالتُّرَّهات) يغرق الطالب عندنا فيما يُشبه ما في كتاب الجغرافيا للصف الأول الثانوي في مصر: أطنانٍ من المعلومات و الأرقام عن ترتيب دول العالم من حيث إنتاج الماشية و الأرز و بقية المحاصيل، و نِسَبِهم المئوية من

1هذا الحديث المنسوب لرسول الله صلي الله عليه و سلم فيه كلامٌ يطول بين علماء الحديث عن صحته، فيُرجَي التنبه إلي هذا.

ذلك الإنتاج العالمي ! و كأن الطالب سيخرج من المدرسة ليُصبِح وزير الزراعة مُباشرةً ! فتكون النتيجة المُتوقَّعة هي أن الطالب صار يكره العلم و التعلم كره الطاعون.

أما في البلاد التي تحترم آدمية البشر فبدلاً من التلقين و الإعتماد علي كمية المعلومات التي حفظها الطالب يعتمدون علي تنمية مهارات التفكير و البحث العلميين، و يعتبرونهما أهم من كمية المعلومات التي يسكبها الطالب علي ورقة الإمتحان ثم ينساها فور خروجه من اللجنة.

فتحوّل الطالب عندنا إلي ما يُشبِه الحمار الذي يحمل أسفاراً لا يفقه ما فيها و لا يهتم به، و يهمه أكثر ما يهمه مجموعُه الذي سيحصل عليه في النهاية حتي يتأهل لكليةٍ من كليات "القمة"، و التي بدورها تقضي علي البقية الباقية من مَلَكات النقد و الإستقلالية و حب البحث العلمي عنده، ليتحول إلي حمارٍ كاملٍ بدون أدني نقص.

و تحوَّل الطالب الجامعي عند الآخرين إلي ماكينة علم لا تَرحم؛ فما بين كُلِّيته التي يتعايش فيها عملياً مع ما يَدرسه و يَري فيها مُدرِّسين ذَوِي كفاءةٍ و مِرَاس، و بين مُدوَّنته/موقعه الشخصي/حسابه في المنتديات العلمية: يعيش جواً علمياً مُكتَمِل الأركان، و لا ينقصه سوي أن يفرّغ نفسه لحضور بعض المُحاضرات و المُؤتمَرات التي تُنظِّمها الشركات الرائدة تقنياً في العالم، و التي يُوجَد منها في بلده و علي مسافةٍ قريبةٍ منه ما ليس بالقليل، و أن يُشارِك في المُسابَقات العالَمية التي تُنظِّمها تلك الشركات.

و الناتج النهائي أن الطلبة عندهم يُخرِجون مُنتَجاتٍ تقنيةٍ من أفضل ما يكون، و بعض مُنتَجاتهم يكون من النوعية التي تقلب كافة الموازين، مثل نواة اللينوكس linux kernel التي بدأها الطالب لِينُوس تُورفالدز linus torvalds ثم ساعده فيها جمعٌ غفيرٌ من المُبرمِجين المُتحمسين في أنحاء العالم، و بيئة سطح المكتب KDE التي بدأها الطالب ماتثياس إترش matthias Ettrich !

بينما رأيتُ و سمعتُ الكثير من المخازي عن طلبتنا، و من أفدحها حينما أراد أحد المُعِيدين أن يمزح مع الطلبة فسألهم سؤالاً سخيفاً: "لو وقعت الفلاشة علي الأرض هل ستنتقل لها فيروسات ؟"، ليُفاجأ المسكين أن الإجابات التي حَصَل عليها كانت من النوع النووي؛

فقد أجاب كثيرون أنهم "لا يعلمون" !

بينما قال البعض الآخر أن "الفيروسات ستنتقل للفلاشة بلا ريب" !

و لله الأمر من قبل و من بعد.

***

و لأننى أَعُدُّ نفسى من أهل البحث العلمى عامةً، و من أهل علوم الحاسب خاصةً: فإن هذا الواقع لا يُرضِينى البتة، و أجد أنه ليس من الدين و التقوى فى شئ أن أظل (بينما أنا قادرٌ مع مَن هم مثلى على تغيير الواقع) مكتوفَ الأيدى أمام كل هذا العبث، فعلي الأقل يُمكِننا فَتْح باب التغيير و لو في تخصصنا العلمي فقط.

فما كان منى إلا أن فكرتُ فى النهضة العلمية للأمة:

كيف يُمكِن أن تبدأ ؟

و إلى أين يُمكِن أن تسير ؟
و أى المسارات يُمكِنها أن تَتخذ ؟
و مَن مِن الناس يُمكِن أن يُساعِد على هذا ؟
و ما هي العقبات التي ستقف في وجه تلك المسيرة ؟
و كيف يُمكِن التعامل معها و التغلب عليها ؟

شَغَلنى هذا الأمر لفترةٍ طويلةٍ حتى تَيقنتُ من تفكيرى و من خبرتى (الصغيرة فى مجال علوم الحاسب) أن واحداً من أيسر السُبل التى يُمكِن للأمة أن تبدأ بها النهضة العلمية الجديدة هو طريق علوم الحاسب البرمجية؛ فهذا النوع من العلوم لا يَحتاج من الإنفاق المادى مُقارنةً بباقى العلوم إلا أقل القليل، و لا يَحتاج إلا الكثير من الفكر و الإبداع، و فى هذا فإن أمم الأرض و شعوبها يتساويان إذا ما تم تحصيل العلم على قدرٍ متساوي.

ثم إن هذا العلم مطلوبٌ فى كل دراسةٍ علميةٍ أخرى سواءٌ أكانت تتعلق بمجال الحوسبة ام لا، و بإنتاجنا للتطبيقات البرمجية الإسلامية فإننا نفتح الباب أمام طوفانٍ من الدراسات التى تستثمر هذه المُنتَجات البرمجية فى باقى المجالات.

و الأهم أن خطوة ترجمة و إنتاج علم من العلوم باللغة العربية ستُحقِّق الكثير من الأهداف (إضافةً إلى ما سنحصل عليه من مُنتَجاتٍ صنعناها بأيدينا و جهدنا)، مثل إثبات أنه ليس هناك تلك الأسطورة المُسمَّاة بـ(لغة العلم الوحيدة) و بالتالى تصلح كل اللغات للإنتاج العلمى و منها اللغة العربية.

و سيُشجِّع هذا كل من كان مُتردِّداً فى التأليف و البحث العلمى باستخدام اللغة العربية على وُلوج ذلك السبيل، ليَنشأ بالتدريج مُجتَمعٌ من العلماء المسلمين يُترجِم و يبحث و ينُتِج باللغة العربية، و كلما كَبُر حجم هذا المجتمع زاد أمل الشباب فى انتشار فكر النهضة و القيام من جديد، و كلما زادت جذوره قوةً و امتداداً فى الأرض الإسلامية، حتى يأتى اليوم الذى يُصبِح التفكير العلمى فيه أمراً مُعتَاداً كما كان فى عصر أجداد الأجداد من العلماء المُسلِمين، حينما كان التاجر و الحَمَّال بل و حارس الدواب عالماً جليلًا تُقطَع إليه الفَيَافِى و القِفَار لطلب علمه و التعلم على يديه.

و كلما كَبُر حجم هذا المُجتمَع الطيِّب زادت اللُّحمة بين أهله من المُسلِمين و ذابت الفوارق العِرْقية بينهم، فلن يكون هناك مصرىٌّ أو باكستانىٌّ أو أمريكىٌّ أو روسىٌّ أو يابانىٌّ أو صينىٌّ أو غيرهن من الجنسيات، بل ستكون هناك الهوية الإسلامية المُوحِّدة التى تَربط بين كل هؤلاء، كما نَرى بين معاشر الملتزمين فى كل عَصْرٍ و مِصْر.

و بكثرة الحديث عن هذه النهضة الإسلامية و العلم الذى يُنتِجه المسلمون سوف تتوحد الجماهير الإسلامية رُوَيْداً رُوَيْداً تحت راية الدين من جديد، و سوف يعود الإحساس بالدين كعنصرٍ فاعلٍ فى الحياة اليومية كما كان عليه فى الأيام الأُوَل، و بالتدريج سيسود الإلتزام بتعاليم الله تعالى و شرعه فى كل مناحى الحياة، و هو الهدف الأعظم الذى نسعى كلنا خلفه و يجب أن يَنْذِر الجميع حياتهم له.

كما أن هناك الكثير من الخبراء و أهل الإبداع فى العالم الإسلامى فى هذا المجال، فهو منتشرٌ الآن جداً على العكس من مجالاتٍ أخرى (مثلاً: تقنية النانو nano technolog، و تقنيات إنتاج عَتَاد الحواسيب computers hardware)، أى أن عَامِل الموارد البشرية متوفرٌ جداً و بدمجه بعَامِل قلة التكلفة: نرى أن كل العوامل أصبحت مُتوافرةً لدينا للنهضة البرمجية التى نرغب بها.

على أن هذه النهضة لكى يُكتَب لها البقاء و الحياة و إتيان ثمارها المرجوة منها: يجب علينا أن نُخطِّط لها جيداً من كل النواحى، عَمَلاً بالمبدأ الإسلامى "**اعْقِلْهَا وَتَوَكَّلْ**"[2]، فيجب بدايةً أن:

- نُحدِّد الأهداف المُستقبَلية المرغوبة بالتفصيل، ثم:
- نُحدِّد ما يجب علينا عمله بالضبط لكى نُحقق هذه الأهداف على أرض الواقع فى قائمة أعمالٍ كاملة، و علينا كذلك:
- تحديد الأولويات بين هذه الأعمال، بأن نُرتِّبها زمنياً بحيث يُتاح لنا أن نرى تقدمنا أثناء العمل، و ما تَبَقَّى فِعله فى المرحلة الحالية، و ما سيتوجب علينا فى المرحلة القادمة.

و لكن علينا فى بداية كل مرحلةٍ من المراحل (و قبل أن نبدأ بالعمل الفِعلِى فيها) أن ننظر إن كانت هناك مُحاوَلاتٌ سابقةٌ فى هذا الشأن يُمكِن أن نستفيد منها فى عملنا؛ حتى لا نُحَمِّل أنفسنا عبئاً زائداً فنخسر وقتاً و جهداً قيِّمين فى إعادة اختراع العجلة، فإن وَجدنا أن هناك مُحاوَلاتٍ سابقةً بالفعل: أصبح لِزاماً علينا أن ننظر لها بعينٍ ناقدة، من حيث: قابليتها لإفادتنا، و كونها على المُستوَى المرغوب أم لا، و هل يُمكِن استعمالها حتى كمرحلةٍ وُسطَى للوصول إلى المُستوَى الذى نرغب فيه.
و النقد هكذا أمرٌ ضروريٌ و ليس ترفاً فكرياً و مضيعةً للوقت كما قد يَظن البعض، و الحق أن الكثيرين سَيَرون أن هذا الأمر مضيعةٌ للوقت و الجهد، و أنه من الأفضل البدء من الصفر بدلًا من البحث عما فعَله الآخرون و مُناقَشته و نقدِه. و لكنى أنبههم أن بعض الجهد فى هذه المُناقَشات قد يُوفِّر علينا الكثير من الوقت فى المُستقبَل نُضيعه فى إعادة صُنع المصنوع، فنستفيد منه نحن فى إنجاز خطواتٍ أخرى تُقربنا من أهدافنا أكثر، فالحق أنه لدينا الكثير من الأمور التى يجب علينا إنجازها كما سأفَصِّل فيما بعد و لذلك فإن أى توفيرٍ للجهد و الوقت يُعد أمراً مطلوباً جداً لنا، فإن لم نَجد فإننا ندعوا الله تعالى بأن يُعيننا على مشوارنا الطويل.

2عن أَنَس بْنِ مَالِكٍ قَالَ: (قَالَ رَجُلٌ:" يَا رَسُولَ اللَّهِ: أَعْقِلُهَا وَأَتَوَكَّلُ أَوْ أُطْلِقُهَا وَأَتَوَكَّلُ" قَالَ:"اعْقِلْهَا وَتَوَكَّلْ "). قَالَ أَبُو عِيسَى وَهَذَا حَدِيثٌ غَرِيبٌ مِنْ حَدِيثِ أَنَسٍ لَا نَعْرِفُهُ إِلَّا مِنْ هَذَا الْوَجْهِ وَقَدْ رُوِيَ عَنْ عَمْرِو بْنِ أُمَيَّةَ الضَّمْرِيِّ عَنِ النَّبِيِّ - صلى الله عليه وسلم- نَحْوُ هَذَا.
و هَذَا الحَدِيثُ أَخْرَجَه "الترمذي" برَقْم (4415)، وحَسَّنه "الألباني" في (تَخْرِيجِ مُشْكِلَة الفَقْر و كَيْفَ عَالَجَهَا الإِسْلَام) - (1 / 23).

فإذا ما وَجدنا بالفعل شيئاً قد أُنتِج من قبل و يُمكِننا الإفادة منه فإننا حينئذٍ سنُحاول ضم صاحب المُكوِّن إلى مجموعتنا العلمية؛ و ذلك لأنه أكثر الناس علماً بالمُكوِّن المَعْنِى و سيُوفِّر علينا وقتاً كبيراً حينما ينضم إلينا ليعمل على تطوير مُكوِّنه؛ فيتفرغ الباقون للعمل على باقى الأشياء التى تحتاج إلى الإنتاج فى المرحلة الحالية. فإن كان هناك ما يَمنع حدوث هذا فإننا سنقوم بتفريغ أنفسنا لتطوير المُنتَج المقصود، و هى حالةٌ أفضل من البدء من الصفر على كل حال.

فإن لم نجد فالبداية من الصفر ستكون هو الحل الوحيد أمامنا، و لن يكون هذا دافعاً للإحباط لدينا على أية حال؛ فالأمر الطبعى أن المشوار الذى سنقطعه ليس طريقاً مفروشاً بالورود تملأه الحماسة و الضياء كما قد يَظن بعض الشباب مِمّن يُعميهم الحماس عن رؤية الواقع من حولهم، فهناك من المُعرقِلات الكثير و الكثير.

و لن يكون غض النظر عن هذه المُعرقِلات و عن أخذها بعين الإعتبار أمراً حسناً، بل سيكون حجر عثرةٍ يُوقِف المسيرة و يَدق عنق مَن لَم يَنتبه إليه، لذا فالحكمة كل الحكمة فى التنبه لكل الأخطار و النظر لها بجديةٍ و التحسب لها بكل ما يُمكِن من أساليب؛ ففى مشوارٍ صعبٍ طويلٍ كمشوارنا لن يكون هناك الوقت أو الذهن الشاغرين عندنا لترك ما بأيدينا من مهامٍ لحل هذه المشكلة أو التحسبِ لعواقب تلك أثناء العمل الفِعلي.

و الوقت الذى يُمكِننا أن نُرتِّب فيه أوراقنا هو الوقت الأول فى عمر المسيرة؛ ففيه سيكون الجميع مازالوا على بر العمل و لم ينخرطوا فيه بعد، و سيكون من المُمكِن التنبه للمشاكل و وضع العدة لحلها قبل أن تطفو على سطح الواقع، و فى نظرى فإن التجهيز الأَوَّلى لتلك المشاكل ينقسم إلى النقاط التالية:

- التوضيح الكامل المُفَصَّل لكل الأمور التى تتعلق بالمسيرة من قريبٍ أو بشكلٍ مُباشر، بل لو كانت هناك بعض الأمور التى تتعلق بالمسيرة بطريقةٍ غير مباشرةٍ و كان هناك الوقت الكافى للتحدث عنها: فإن العقل السليم يَحُض على حسم أمرها فى البيانات الأولى من زمن الإعلان عن الحركة العلمية.

- الرد الإستباقي على أية اعتراضاتٍ نرى أنها ستخطر على بال من يَسمع عن المسيرة الجديدة فى أى جانبٍ من جوانبها، و إن كانت هناك اعتراضاتٌ موجودةٌ بالفعل من قبل: فالرد عليها بكل وضوحٍ يجب أن يَسبق الرد على الاعتراضات التى يُتوَقع صدورها فيما بعد، و هنا يجب ألا نخشى التكرار فى الرد على الاعتراضات، أى أن بعض الاعتراضات ربما يكون قد تم الرد عليه بشكلٍ غير مباشر أثناء الحديث عن المسيرة (فى مواقع غير موقع الرد) و لكن هذا لا يعنى أنه يُمكِن الإكتفاء بما قِيل من قبل، بل يجب أن يتم الردّ بشكلٍ مُباشرٍ على الاعتراض و لو كان فى هذا بعض التكرار و الإطناب؛

فهناك مِن الناس مَن لا يُمكِنه التنبه إلى أن الردود على الإعتراضات تكمن فى الشروح السابقة،
و هناك من لا يَقتنع إلا بالكلام المُباشر الصريح و ليس الكلام الذى يَجمع بين أكثر من هدفٍ فى ثناياه، و غيرهم و غيرهم،
فلكل هؤلاء يجب أن تكون الردود على الاعتراضات مُفرَدةً فى مكانٍ واحدٍ و غير مُتناثِرةٍ هنا و هناك حتى يرى الأمور واضحةً أمامه.

- التنبه إلى الجوانب القانونية و/أو الدينية و/أو الإجتماعية إن كانت موجودةً، و أخذ الحيطة لها و دراستها بتأني؛ لأن مثل هذه الجوانب يكون لها أخطر الآثار فيما بعد عند عدم أخذ الإحتياط الواجب لها.

- التنبه إلى الجانب التمويلى و الإدارى من المسألة؛ فمشروعٌ نهضوىٌ كبيرٌ كهذا المشروع الذى نُريده يجب أن يُدار بكل حكمةٍ و دقةٍ حتى لا يكبو لأخطاءٍ إداريةٍ كان يُمكِن تلافيها، فنخسر كل شئ، و خاصةً و أن هناك من سيترصد له و يقف فى وجهه عنوة. و كذا فإن الجانب التمويلى من أخطر الجوانب فى المسألة و ينبغى حسمه منذ البداية بكل وضوح.

و سيراً على هذا النهج الذى نبَّهتُ إلى أهميته سيكون سيرنا فى مشوارنا، و ما هذه الرسالة المُبارَكة إلا المثال العملى لكل ما أوردتُه؛ فقد كُتِبت لتكون مرجعاً شاملاً شافياً يُغطِّى كل الجوانب الهامة التى تخص المشروع العلمى الجديد الذى أرغب فى طرحه و تنفيذه على أرض الواقع.

عن المُستقبَل

كانت الخُلاصة فيما وصلنا إليه فى مقدمة هذه الرسالة أن الأمة الإسلامية تحتاج إلى نهضةٍ علميةٍ حقيقيةٍ تكفل لها استرداد مكانتها بين أمم الأرض، على شرط أن تكون تلك النهضة قد تم الترتيب لها بشكلٍ عميقٍ يُغطِّى كافة النواحى المتعلقة بها، مع وضع جدولٍ زمنيٍ يُوضِّح الخطوات التى يجب أن تسير عليها فى تالى أيامها فلا تكون عُرضةً للتخبط و التشتت.
و أن يقوم على أمر هذه النهضة من يمتلك علماً جيداً بمجالها، فيضع لها جدولها الزمنى و يبنى الأساسيات الأولى لها ثم يدعوا الناس لها بعد الوصول فيها إلى مرحلةٍ ليست بدائية، ثم يقوم ذلك الشخص بالتعاون مع من انضم إليه بتنسيق العمل فيما بينهم لتحقيق الجدول الزمنى على أرض الواقع.
هذا إن ظل الجدول الزمنى كما هو بعد انضمام المُنضمِّين الجُدد إليه؛ حيث أن انضمام الكثيرين من أصحاب مجالٍ معينٍ كان قد وُضع فى ترتيبٍ مُتأخرٍ فى التنفيذ قد يُغرى بتقديم تنفيذ ما يتعلق بذلك المجال زمنياً لتَوفُّر القوة البشرية الخاصة به، و غير هذا كثير.

و ذَكَرنا فيما سبق أيضاً القول بأن أكثر العلوم قابليةً للزرع فى الأرض الإسلامية هى العلوم البرمجية؛ لما تتمتع به من مُتطلَّباتٍ ماليةٍ أقل من التى تتطلبها باقى العلوم الأخرى، و خاصةً بعد نهوض حركة المصادر البرمجية الحرة التى وفَّرت لكل من هم على شاكلتنا ما يلزمهم من أدوات النهضة البرمجية بدون أى مقابلٍ مادى.
فتُوجَد أنظمة تشغيلٍ operating systems و مُترجِماتٌ compilers و مُفسِّراتٌ interpreters و مُنقِّحاتٌ debuggers و غيرهن من الأدوات من المُندرِجات تحت تصنيف

البرمجيات الحرة free software، و يُمكِننا أن نستخدم كل ذلك للوصول إلى الهدف الذى نرغبه من تحقيق النهضة العلمية بحيث لا تُكلِّفنا إلا أقل القليل مادياً حتى لا يُزعجنا أمر التمويل. و كذا لتوافر العنصر البشرى من المُبرمِجين و الخبراء فى مختلف المجالات البرمجية (و إن غلب عليها المجالات ذات الطابع التجارى).

و ما دام الأمر هكذا فإننى أجد أن الوقت مُناسبٌ للدعوة إلى مشروع أُريد من الجميع المشاركة فيه معى لتحقيق ما نتمناه لأمتنا و ديننا علمياً. فالمشروع الجديد الذى أطلقت عليه اسم (البرمجة بإبداع) هو النموذج الذى وضعتُه لكل ما تحدثت عنه فى أمر النهضة البرمجية الإسلامية.

و خلاصة المشروع و الغاية منه هما: تزويد الأمة الإسلامية بكل الأساسيات التى تخص علوم الحاسب البرمجية باللغة العربية و على كل المُستوَيات، بدءاً من ترجمة أهم المنشورات التى تخص أساسيات مجال البرمجيات، وُصولاً إلى إنتاج لغات البرمجة العربية الحرة المُختلفة التى تُنافِس اللغات الأجنبية فى قوتها و نُضجها، و ليس انتهاءاً بصُنع نظم التشغيل العربية الحرة.

غايات المشروع:

و تفصيل غايات المشروع فيما يلي:

- ***ترجمة و تأليف المُؤلَّفات الخاصة بالعلوم البرمجية الأساسية إلى اللغة العربية***، و أقصد بالعلوم الأساسية: تلك العلوم المُتعلقة بـ: تصميم لغات البرمجة، و إنتاج المُترجِمات و المُفسِّرات و المُنقِّحات و كافة الأدوات البرمجية الخاصة بها، وُصولاً إلى إنتاج بيئات البرمجة المتكاملة integrated development environments، بل و وصولاً إلى ترجمة كل ما يتعلق بتصميم و إنتاج نظم التشغيل المختلفة.

  و لا أعني هنا الترجمة من اللغة الإنقليزية فقط، بل يجب الإهتمام بترجمة كل المُؤلَّفات الهامة التي تختص بالمجالات المذكورة بالأعلي (أو أي مجالٍ نهتم به فيما بعد) و إن لم تكن باللغة الإنقليزية. فلو كانت هناك رسالة دكتوراة أو ماجستير متميزة و كانت افتراضاً باللغة الألمانية فيجب أن يتم ترجمتُها. إلا أنه نظراً لأنه من المُعتاد أن يقوم المُتخصصون في المجال التقني باستخدام اللغة الإنقليزية فلا أظن أن مثل هذه الحالة سوف تحدث كثيراً.

  كما أن الوصول إلي مرحلة تتبع الأبحاث المُتميزة علمياً بلغاتٍ غير الإنقليزية سوف يأتي في مراحل غايةٍ في التقدم في المشروع، و حينما تترسخ أقدامه و يكون له مصادر تمويله الخاصة و كوادره العلمية المُتفرِّغة.

- ***إنتاج مُنتَجاتٍ عربيةٍ تخص الأدوات الرئيسة فى المجال البرمجى***، بدءاً من لغات البرمجة العربية بمُختلَف أنواعها و انتهاءاً بنظم التشغيل العربية بمُختلَف أنواعها.

الدوافع:

و الأسباب وراء ذلك هى:

- **عامل تقويةٍ للغة العربية:**

و ذلك باستخدامها فى مجال العلوم البرمجية بتوسع بالغ. و الغاية من ذلك جعل العربية جزءاً لا يَتجزأ من العلم البرمجى؛ لأن الإعتماد على الترجمة بدون نقل العلم نفسه إلى اللغة و الإسهام فيه باستخدامها ليس بالأمر الذى يحفظ اللغة من الإنهيار، بل يؤخر حدوث ذلك فقط، و ما نراه من تفضيل الكثيرين لقراءة الكتب الإنقليزية على الكتب العربية (و إن كانت تلك الكتب العربية أفضل من حيث الشرح و كمية المعلومات) يَرجع أول ما يَرجع إلى النظرة الإنبهارية التى ينظرون بها إلى الكتب الإنقليزية عامة، و النظرة الدونية التى ينظرون بها إلى الكتب العربية عامة، بِغَض النظر عن المُحتوَى و المُستوَى العلمى.

و يرجع هذا بدوره إلى الهزيمة النفسية التى يُعانون منها؛ فهم يَرون أن العلم ينبع حيث ينبع من أراضٍ غير عربيةٍ و بأيدٍ غير عربيةٍ و بلغةٍ غير عربية، فمن الطبعى هكذا أن ينظروا للعربية على أنها لا تَصلح وعاءاً للعلم، و أن ينظروا للغة الإنقليزية التى يُكتَب بها أهم العناصر المُمثِّلة للبرمجة (و هو الكود نفسه) على أنها لغة العلم الحقيقية، و الكتب التى تُكتب بلغة العلم لابد من أن تكون هى الأفضل علمياً !

و رغم أن الكثيرون الآن يرون أن الترجمة و تأليف كتب الشروحات العلمية باللغة العربية أمرٌ جميل (و يشهد على انتشار هذه النظرة كميةُ الكتب التى يُؤلِّفها العرب بالعربية، و التى أصبحت تُغطِّى مجالاتٍ علميةٍ برمجيةٍ كثيرةٍ بالفعل و بكمياتٍ ليست بالقليلة)، أقول: رغم هذا فإنه كما أسلفنا القول لا يكفى للحفاظ على اللغة العربية كما يفعل إنتاج العلم من أساسه بها؛

فنحن حينما نُترجِم نحتاج دائماً إلى إخبار القارئ بالمُرادفات التقنية الأصلية فى الإنقليزية حتى يستطيع معرفة معنى المُصطلح الإنقليزى متى ما رآه، و بالتالى يكون من السهل عليه قراءة و فهم مُحتوَى الكتب الإنقليزية، كما أنه لا بد له من مُطالعة تلك الكتب الإنقليزية و لو بعد حين، بل و لا بد له من التوسع فى ذلك الإضطلاع إذا ما أراد مُتابعة الجديد أولاً بأول، و كذلك إذا ما أراد التعمق فى جزئيةٍ من البرمجة لم يهتم بها أحدٌ من المُؤلِّفين بالعربية من قبل، أو كان الاهتمام بها ضعيفاً و لا يُعطيه ما يحتاجه من تعليم مُتعمِّق أو مُحدَّث.

أما حينما تُنتِج العلم بلغتك و تُطور فيه مُستخدماً إياها فإنك تكون حينئذ قد رسَّخت أقدام لغتك فى المجال البرمجى بأقوى السبل؛ لأنك حينما تُنتِج علماً بلغتك و تُوثِّقه بها لن يكون على الراغب فى تعلمه من بنى جلدتك استخدام لغةٍ أخرى فى مُطالعته و/ أو نقده و/أو تطويره، و بالتالى ستكون كل الألفاظ التى تُستخدم من ثوب اللغة نفسها لا رقعاً من لغةٍ أُخرى تُلصَق بلغتك لصقاً.

- **تنشئة أجيالٍ مُسلمةٍ و/أو عربيةٍ قادرةٍ على البرمجة منذ الصغر:**

لوجود لغة برمجةٍ بلغة دينها و/أو لسانها، و الفارق كبيرٌ للغاية بين أن يتم تدريس البرمجة للأطفال بلغتهم مع البرمجة بلغة برمجةٍ أجنبيةٍ (إنقليزيةٍ بطبيعة الحال) و بين أن يكون التعليم و اللغة البرمجية نفسها عربيين مئة بالمئة.

و أنا أقول هذا الكلام و أنا من الأجيال التى عانت و تُعانِى حتى الآن من جَرّاء كون العلم البرمجى ليس علماً أصيلًا فى الثقافة العربية، و كل ماله علاقةٌ بالإحتراف فى المجال البرمجى هو من إنتاج غير المسلمين/غير العرب و باللغة الإنقليزية (إلا فيما ندر)، و ليس للعربية نصيبٌ من التأليف العلمى إلا القليل، و ما كان من ذلك التأليف إضافةً حقيقيةً إلى العلم هو أقل القليل.
لذا فأنا و جيلى كله (كما الأجيال السابقة) مُجبرون على تحصيل العلم من مَظَانّه بلُغةٍ غير لغتنا، و لا أخشى أن أقول أن الأمر لم يكن سهلًا أو مُبهجاً على الإطلاق بالنسبة لى؛ فقد كان على أولًا أن أعتاد على اللغة الإنقليزية كلغة تحصيلٍ علمى، و أن أحفظ من تعبيراتها العلمية و مُصطلحاتها التقنية ما أقدر عليه، و حتى بعد أن فعلتُ هذا كان على أن أُعَوِّد نفسى على قراءة الكتب الإنقليزية لفتراتٍ طويلةٍ حتى أُحَصِّل من العلم أقله؛
فالواقع أن الساعة الزمنية التى تكفينى لمُطالَعة كتابٍ عربىٍ كبير الحجم و فهمه فهماً شبه كاملٍ لا تكاد تكفينى لمُطالَعة فصلٍ واحدٍ فى كتابٍ إنقليزىٍ سهل اللغة قريب الأسلوب، فما البال بالكتب التى ما ألَّفَها مُؤلِّفوها إلا و كان التحذلق اللغوى فوق أطراف أنوفهم ؟!

و كلما وَجدتُ نفسى فى موقفٍ سخيفٍ يَفرض علىّ أن أُطالِع كتاباً إنقليزياً لغته (بالنسبة لى أنا المغلوب على أمرى) كطلاسم السحرة: كرهتُ الوضع الحالى أشد الكره، و كرهتُ أكثر منه مَن كانوا يستطيعون التغيير فيه و لم يفعلوا (بل و ربما حتي لم يُحاوِلوا)، و كرهتُ أكثر من كل هذا من ثَبَّطوا همم من حاول التغيير و كانوا سبباً فى فشلهم بشكلٍ مُباشرٍ أو غير مُباشر.

و عن تجربةٍ حقيقيةٍ أقول أن العلم إذا أصبح بلُغتنا كان بإمكاننا الإبحار فيه بمنتهى البساطة و الوضوح، و سيكون تركيزنا على التحصيل الحقيقى للعلم و لن نُضيِّع الجهد و الوقت الغاليين فى تحصيل أدواتٍ (مثل اللغة الأجنبية) ليست مرغوبةً فى حد ذاتها، بل إن الهدف منها فقط أن تُمكِّننا من تحصيل العلم نفسه.
و سيكون الأمر بالنسبة للصغار أسهل من شرب الماء القراح؛ فنحن حينما نريد تعليم الطفل الصغير البرمجة بلغة برمجةٍ أجنبيةٍ سيكون علينا أولًا: أن نجعله يفهم أن هذه لغةٌ أخرى كلغتنا العربية، و لكن يُمكِن للحاسوب فهمها إذا ما كُتِبت على نحوٍ معينٍ و بألفاظٍ مُحدَّدة، و بعد عناء إيصال ذلك المعنى له (و هو الأمر الذى يكاد يكون مُستَحيلًا على الأطفال غير الإنقليز) فسيكون أمامنا أن نجعله يعى معنى كل كلمةٍ إنقليزيةٍ (تُستخدم فى لغة البرمجة) فى الحياة العادية و معناها فى البرمجة حتى يستوعب الفكرة جيداً (و هو الأمر الأقرب للمُستَحيل فعلياً على الطفل الصغير)،

و بالتالى لا يكون أمام من يُعلِّمه إلا قول (اكتبها كما هى) و (احفظها و انته) و (ستعرف معناها فيا بعد و ليس الآن).

أما حينما نستخدم لغة برمجةٍ عربيةٍ فإن الأمر لن يُكلِّفنا إلا قول جُملةٍ واحدةٍ: (يا صغيرى: الحاسب لا يفهم إلا تعبيراتٍ مُعيَّنةٍ هى كذا و كذا و كذا).
و بالطبع لن نحتاج إلى شرح المعنى الحياتى لهذه التعبيرات؛ لأن الطفل: يستعملها بالفعل، أو يسمعها، أو هى قريبةٌ للغاية مما يَستعمل أو يَسمع. فالجُمَل الشرطية كجملة لو (التي سيلي شرحها في أثناء شرح قواعد اللغة) على سبيل المثال لا تَحتاج إلى أى شرحٍ كجملة if التى نحتاج لإفهام الطفل معها معنى كلمة if و else أوّلا.
و هكذا يُمكِننا التركيز على تعليم الأطفال فكرة البرمجة لا تحفيظهم الكلمات المحجوزة الخاصة بلُغة البرمجة. و سنرى أن الأطفال سيتسابقون (فيما بعد) فيما بينهم بالألغاز البرمجية و حلولها، على العكس من الحالة الثانية التى سيتسابقون فيها بحفظ التلخيصات و أمثلة البرامج و القواعد لسكبها على ورقة الامتحان ثم نسيانها تماماً بعد ذلك !

- **إعطاء الفرصة لربط المسلمين غير العرب باللغة العربية بشكلٍ أكبر:**

من خلال تحويل العلوم البرمجية إلي الغة العربية فتُصبح اللغة العربية بالنسبة لهم لغة دينٍ و تخصصٍ علمي في ذات الوقت، و هو المدخل المُتميز لربطهم بالدين بِصِلةٍ و ثيقة؛ حيث أن المسلمَ غير العربي على كاهله من الهموم الكثير، مما يشغله عن تحصيل العلم الشرعى و تعلم العربية (لغة الإسلام) إذا كان من أهل العلوم الحياتية؛ حيث سيكون عليه أن يُحَصِّل اللغةَ الإنقليزية (اللغة الأكثر انتشاراً فى مجال العلوم الحياتية) و العلمَ الحياتى نفسه، إضافةً إلى جهده الذى يصرفه فى وَصْل رحمه و بقية أمور حياته الخاصة، فلا يتبقى له من الوقت شيءٌ للإهتمام بعلوم الشرع إلا على أخف الأوجه و أشدها ضرورة.

و لو أننا حَوَّلنا العلم البرمجى إلى علمٍ أصيلٍ فى اللغة العربية و جعلنا لها قدماً راسخةً فيه لكان بإمكان المسلم غير العربى استغلال ذلك أفضل الاستغلال بتعلم اللغة العربية واستخدامها كلغة تحصيلٍ علميٍ للعلم الحياتي و كلغة دينٍ، فلا يخسر من الجهد شيئاً و كما يقولون (يصطاد عصفورين بحجرٍ واحد).

و هذا الهدف يُعَد مِن أول الأهداف التى يجب أن يَسعى لها كل راغبٍ فى النهضة بالأمة من جديد؛ فليس المسلمون كلهم من العرب، بل إن أغلبهم من غير العرب، و على أكتافهم كما على أكتاف المسلمين العرب تقع مسؤولية النهضة بالأمة من جديد، و ما دامت اللغة العربية تُشكِّل أمراً ضرورياً لا يتجزأ من الحياة الإسلامية فيجب أن يُبذَل أقصى الجهد فى سبيل تقويتها عند غير

العرب و جعلها لهم كما كانت بالنسبة لمُسلِمى الزمان الأول من العجم كالإمام محمد بن إسماعيل البخارى (صاحِب: صحيح البخاري) رحمة الله عليه قضيةً غير قابلةٍ للنقاش أو الجدال حولها من الأصل ناهيك عن التخلى عنها.

- **عاملٌ نفسيٌ هامٌ للغاية:**

يجعل من الحديث عن النهضة الشاملة و استرجاع مكانتنا الحضارية فى كل الميادين الحياتية أمراً قوى الحجة إلى أقصى الحدود؛ فما دام الأمر قد نجح فى علم من العلوم (و خاصةً إذا كان علماً في غاية التطور مثل علوم الحوسبة) فهو قابلٌ للنجاح فى كُل العلوم. فقد أُثبِت أنه لا لغة للعلم و أن المهم هو الإنتاج العلمى لا اللغة التي يتم بها ذلك الإنتاج، على العكس مما يقوله دجالو هذا العصر من أن العلم: غربى الهوية، إنقليزى اللغة، و أن تغيير ذلك من المُحال.

و بدلاً من المجال البرمجى فقط سنجد أن الإنتاج العلمى بالعربية قد توغل فى باقى الميادين العلمية بدون استثناء، و رُوَيْداً رُوَيْداً سيأتى الزمن الذى يَستنكر طالب العلم فيه أن يَطلب العلم بِلُغةٍ غير اللغة العربية، و سيقول رداً على هذا كلمةً سأبذل الغالى و الرخيص بإذن الله عز و جل لأسمعها من كل الناس "أنا أريد تعلم العلم الفلانى لا اللغة الإنقليزية !".

- **البرهنة على سَفْسَطة القائلين بأن الإلتزام بتعاليم الدين ضد العلم و الحضارة:**

حيث أن الأجيال المُحتَلة عقلياً (ممن خانوا دينهم و باعوا أنفسهم للثقافة الغربية بثمنٍ بَخْسٍ) يستغلون الجهل المنتشر بين أهل الأمة، و الدجلَ و الشعوذةَ اللذين انتشرا فى كل مكانٍ ليُسيئوا للدعوة إلى الالتزام بتعاليم الدين الحنيف أبشع الإساءة.

و دائماً ما يَصب حديثهم فى هذا المَصَب النجس، و لأنهم انتشروا فى كل مكانٍ و تجدهم بسهولةٍ عند كل زاويةٍ و تحت كل حجر، و لأن الجهل انتشر بالفعل بين عامّة الأمة انتشار النار فى الهشيم: فقد لاقت مزاعمهم أرضاً خصبةً و عقولاً تُصدِّق و تُؤمِن. و أصبح الرد على كل من يَأمر بمعروفٍ أو يَنهى عن مُنكرٍ: أن "كُفَّ عن التخلف"، و هو ما أؤمن أنه قِيل لكل مُلتزِمٍ حتى و إن أَمَر بأوضح المعروف و نهى عن أشنع المُنكَر !

و أصبح شوكةً فى حلق كل ملتزمٍ أنه يُوصَم دون أى جنايةٍ أو تَحقُّقٍ بأنه ضد العلم و الثقافة و الحضارة، على الرغم مِن أن أغلبَ مَن فى معاشر المُلتزمين و أغلب رؤوسهم و مشايخهم هم مِن أهل العلم و الفضل فى العلوم الحياتية قبل العلوم الشرعية !، و لكن الكره لا يدع للعدالة جانباً و الجهل لا يدع للتحقيق و التأكد مكاناً.

بل و أصبح من سَمْت أهل العلم و الدراية بالعلوم و الحضارة (عند كثيرٍ من الناس) الإبتعاد فى الهيئة و القول عما يُشابه هيئة و قول أهل الالتزام، و أصبح الناس عندهم قِسمَين: مُلتَحين مُتخلِّفين، و غير مُلتَحِين مُتحرِّرين تقدميين. و القول بخلاف ذلك عندهم هرطقةٌ و جدلٌ فارغان !

لذا فإن مثل تلك الدعوة للنهضة تُعَدُّ بمثابة الرد العملى على تلك الدعاوى البغيضة. فإذا ما أَمَرْنا بعدها بمعروفٍ أو نَهينا عن مُنكَرٍ و اتُّهِمنا بالتخلف: كان بإمكاننا ترك البرهان النظرى و البرهنة بالواقع على العكس. و أن أنفع الناس للأمة هم المُلتزِمون فيها بتعاليم ربهم جل و علا. و أن أشد الناس إكثاراً فى الحديث بدون جدوى و تثبيطاً لهِمَم أهل الحَلّ و العَقْدِ هم مَن نا صبوا أهل الدين العداء و وقفوا فى وجوههم.

- **هدفٌ شخصيٌ صرف:**

وهو رغبتي الذاتية في تحصيل علوم الأساسيات الحاسوبية بشكلٍ عمليٍ إحترافي. و أنا أضع هذا الهدف هنا لأنه من أهم الأهداف التي تُحرِك هذه المسيرة علي الإطلاق. و من خلاله يُمكِن تفسير كثيرٍ من القرارات التي تبدو غريبةً للوهلة الأولي. مثل قرار بناء كل شيءٍ من الصفر و عدم الإستعانة بأي مُنتَجاتٍ أخري كما سيلي ذِكْره.

و كذلك العمل الدؤوب لعامين كامِلين بدون إخبار أحدٍ بحقيقة المشروع (اللهم إلا قِلَّة تُعَد علي أصابع اليد الواحدة بحكم الظروف فقط). و الإصرار علي المُبالَغة في بعض الأحيان فيما يجب: دراسته و/أو عمله و/أو التخطيط له في مُختلَف مراحل المشروع. و في الأوقات القادمة من حياة المشروع سوف يظهر هذا في بعض القرارات الغريبة (مع أنها لن تكون مُستنكَرة).

و دعوني أسترسل هنا بعض الشيء لأنني أُريد إيصال رسالةٍ مُعيّنةٍ إلي أشخاصٍ بعينهم: فبالنسبة لي فإن أبشع كوابيسي هو أن أكون علي فراش الموت في النَّزْع الأخير. و أنظر إلي حياتي الماضية فأجد أنني جئت إلي الدنيا كأي حيوانٍ أعجم: أكلتُ. و شربتُ. و نمتُ. و قضيتُ حاجتي. ثم مِتُّ و بعد موتي أصبحتُ كأني لم أكن !!

إن كابوسي الأشنع هو أن أموت بدون أن يكون لحياتي أي معنيً أو أثر. و لا أقصد الشهرة هنا؛ فكم من أناسٍ أثَّروا في حياتنا بشكلٍ كبيرٍ جداً و لا نعرفهم و لا نذكر أفضالهم. إنما أقصد بالكابوس أن أكون عادياً في ولادتي و حياتي ثم مماتي. أن أكون رقماً مُضافاً إلي تعداد الناس لا يُحدِث فارقاً إن كان بالنقص أو بالزيادة.

ما أريده حقاً (و الشيء الوحيد الذي أرتضيه لنفسي و أجده يليق بي) هو أن أكون علامةً فارقةً في المجال الذي أتخصص فيه.

أن أصير مَرْجِعاً علمياً يسير علي قدمين و يُعَد كلامي من الأدلة التي تفصل بين المُتنازِعين في المسائل العلمية.

بل أُريدُ أن أكون مُقتحِماً لأراضٍ في المجال العلمي لا يَجرؤ علي اقتحامها الكثيرون.

بل أُريدُ أن أكون فاتِحاً لأراضٍ لا يَعلم عنها أغلب الناس (إن لم يكن كلهم) شيئاً.

و من وجهة نظري: فهذا ليس بخارقٍ لطبائع الأمور؛ فما دام المرء بعيداً عن الغرور و الانتقاص من قَدْر غيره فيُمكِنه أن يحلم بما يُرِيد، و أظن أنني أتعلم الدروس التي لابد لي من تعلمها ما دمتُ قد سلكتُ هذا الدرب درساً تِلو الآخر.

و ما يهمنا هنا: أن هذا المشروع هو أفضل السُبُل على الإطلاق لإرضاء طموحي (هل أقول: جشعي ؟)، و يُمكِنني من خلاله تحقيق كل تلك الأحلام التي تكاد تُفجِّر مخي و أعصابي تفجيراً، كما أنه أيضاً فرصةٌ لكل من هم على مثل شاكلتي لكي يُحاوِلوا معي تحقيق أحلامهم على أرض الواقع.

الإعتراضات المُوجَّهة لإنتاج لغة برمجةٍ عربيةٍ و الرد عليها

ما دام إنتاج لغة برمجةٍ عربيةٍ هو الهدف الأول الذي يجب علينا أن نُحقّقه لأجل نهضةٍ علميةٍ برمجيةٍ حقيقيةٍ كما أسلفنا الذِّكْر، فإنه يجب علينا أن نَرُدّ علي أية اعتراضاتٍ موجودةٍ علي هذا الهدف؛ فكما أوضحنا من قبل فإنه يجب علي القائمين علي شأن أي مسيرةٍ علميةٍ ذات تخطيطٍ جيدٍ أن يقوموا بتوضيح كافة ما يتعلق بها منذ البداية و أن يقوموا بالرد علي أية اعتراضاتٍ تُوجَد. و قد شرحتُ أسس مشروع (البرمجة بإبداع) بإيجازٍ من قبل، و حان الآن وقت الرد علي الاعتراضات التي تُواجِه الهدف الأول و الأهم و هو إنتاج لغة برمجةٍ عربيةٍ احترافية.

و قد جَمعتُ هذه الإعتراضات كمُعظَم البيانات التي أعتمد عليها في هذا الكتاب من شبكة المعلومات الدولية، و على الأخص من المُنتدَيات التقنية و المواقع التى كانت تُثَار بها قضية إنتاج لغة برمجةٍ عربيةٍ فيُسهِم الجميع إما موافقةً لهذا الأمر أو رفضاً له.
فكثيرون كانوا يطرحون هذا الأمر على باقى المشاركين فى المُنتدَى أو الموقع إما لمُناقشته من حيث المبدأ فى حد ذاته، أو حتى على سبيل دفعهم للمشاركة معهم فى إنتاج لغة برمجةٍ عربيةٍ بدأوا أو سيبدأون فى إنتاجها بأنفسهم، أو حتى لمُناقشة مُحاولةٍ تتم حالياً سمعوا عنها من غيرهم أو من خلال شبكة المعلومات.

المهم أنه فى كل الأحوال فإن المشاركين فى المُناقَشات لا يخرجون عن إحدى المجموعات التالية:

- **الأولى:**

و هى مجموعة المُتحمسين للأمر بطريقةٍ كبيرةٍ للغاية قد تصل إلى حد التعصب الأهوج، و هو ما قد يدفعهم إلى الإحتدا د الكبير أثناء مناقشة الأمر مع الرافضين له، و أ سلوب خطابهم يشف عن عدم تحملهم لوجو د من يرفض المسألة برمتها أو على الأقل سماع أقوال ذلك الرافض.
و أنا (و إن كنتُ أُوافِقهم فى أ صل المسألة) فإننى فى معظم الأحيان أرفض الإحتدا د الكبير الذى يُصاحِب كلٍ خطاباتهم، و أقول فى **معظم الأحيان** لأن هناك أوقاتٌ لا يكون مُمكِناً بالنسبة لى (مثلهم تماماً) أن أضبط أعصابى و عندها يكون تَهَيُّج الأعصاب وِزراً يقع على كاهل المُتسبِّب فيه و لا يكون علىّ أو عليهم فيه أى ذنب.

- **الثانية:**

و هى مجموعة الرافضين للأمر بطُرُقٍ ربما تصل بدورها إلى حد التعصب، و يُصاحِب أقوال كثيرٍ منهم بدورهم حِدَّةٌ كبيرةٌ و سخريةٌ مريرةٌ أثناء مُناقَشاتهم للمُوافِقين و المُتحمِّسين للأمر، و رغم إ صابة بعضهم فى بعض إعتراضاتهم على الأمر إلا أن الأغلبية منهم لا يُعبِّرون عن إعتراضاتٍ قويةٍ، بل عن إعتراضاتٍ فى قمة الضعف و الهزال، و لا تُعبِّر إلا عن رفضٍ نفسيٍ قاطعٍ منهم للمسألة من حيث المبدأ و محاولة إيجا د أي مُسوِّغٍ منطقيٍ لذلك الرفض !.

- **الثالثة:**

و هم قليلٌ من الناس من المُتوسِّطين فى رفضهم أو قبولهم للأمر و يُحاوِلون مُناقَشة الفرضيات المُختلِفة بعقلانية، و العقلانية هنا لا تعني التخلي عن التحمس لتأييد أو رفض الأمر بل تعني الإنصاف في التناقش و الإعتما د علي البراهين و الأ دلة بحيا ديةٍ تامّة.
و لا أعني بقِلة العد د أن عد دهم الحقيقي قليلٌ بل أعني أن عد د من يظهر منهم في مثل تلك المُناقَشات الحامية هو القليل، كما أن تو سطهم هذا لم يُبرِز أقوالهم فى النقا شات التى دارت و تدور فى هذا الموضوع، بل كانت السيطرة على النقا شات (كالعا دة) لأ صحاب المجموعتين السابقتين لحما ستهم الزائدة التى كانت تدفعهم للكتابة دفاعاً أو هجوماً، و لصوتهم العالي و الأ سلوب الساخر الذي لا يهتم صاحبه لا بدليلٍ و لا ببرهانٍ و الذي أُطلِقُ عليه ا سم (أ سلوب تطليع اللسان).
و المُلاحَظ أن أغلب العرب في نقا شاتهم الدينية و/أو السيا سية و/أو التقنية يكونون أقرب ما يكون لفريقين من الدِيكة المُتَنافِرة، و يكون التعصب عندهم لصالح فريقٍ ضد فريقٍ آخرٍ و إ ستعمال التهكم و السُباب أمراً طبعياً للغاية، و لتذهب البراهين و الحُجَج العقلية إلي حيث ألقت !.

و فيما يلى أطرحُ الإعتراضات الرئيسة التى أظن أن باقى الإعتراضات تتفرع منها، و قد قمتُ بالتنسيق فيها قدر الا ستطاعة بحيث تخرج سليمة اللغة و فى نفس الوقت تُعطِى ذات المعنى و الإحساس اللذان

تُعطيهما كلمات المعترض الأصلية. و لما كنتُ قد فشلتُ فى معظم هذه المُحاوَلات فإن التغييرات التى سيجدها القارئ ستكون طفيفةً للغاية إن وُجِدت من الأصل.

و قد أدرجتُ إعتراضاتٍ تُمثِّل الطيف الكامل لكل الإعتراضات التى قرأتُها فى المُنتدَيات المُختلفة. و لم ألتزِم بأن تكون كلها من الإعتراضات الوجيهة: بل فيها إعتراضاتٌ من النوع الهزيل الضعيف الذى لا يُمكِنني أن أهمل وجوده و الحاجة إلى الرد عليه. و ذلك لأنه ربما يكون (رغم ضعفه) قد صدر عمن هو مقتنعٌ به. و كذلك لمنع خُطورِه على خاطر أحدٍ آخرٍ فيما بعد فنكون قد كفينا أنفسنا مُؤنة الرد عليه مُستقبَلًا.

و من البَدَهِى طبعاً أننى لم أهتم بذكر أسماء أصحاب الإعتراضات المُختلِفة. و محوتُ ما كان منها موجوداً فى صلب الإعتراضات ذاتها؛ لأن الأمر غير شخصيٍ و لا يحق لى أو لغيرى أن يُحوِّله إلى ذلك. و هذه نقطةٌ فائقة الأهمية أرجو التنبه لها جيداً.

***

1- ما معنى أن نبدأ من الصفر فى الوقت الموجود فيه لغات برمجة مفتوحة المصدر ذات قدرات عالية مثل الـjava. لماذا لانشارك (نحن العرب) فى تطوير تلك اللغات مثل آلاف العقول حول العالم بدلًا من الكلام الكثير.
ما نحتاج إليه فعلًا هو عملية تعريب للمراجع الخاصة بالبرمجة.

الحق أن هذا القول يعنى أن صاحبه يتحدث عن أمرٍ مُختلِفٍ بالكُلِّية. فكما أسلفنا القول من قبل فإن النهضة البرمجية التى نرغب فيها ليس الهدف الوحيد الذى نود الوصول إليه من خلالها هو رفع الإسهام العلمى الإسلامى فى مجال العلوم البرمجية فحسب؛ لأنه إن كان الأمر كذلك لكانت المُساهَمة مع مُجتَمع المصادر المفتوحة العالمى هو الطريق الأمثل لنا بالفعل كما أسلف صاحب الإعتراض القول. لكن النهضة لها من الأهداف الأهم الكثير و التى لا تحققها هذه الوسيلة.
و هى كما ذكرناها من قبل: تقوية اللغة العربية فى مجال البحث العلمى. و ربط الأجيال الإسلامية الشابة و الطفلة باللغة العربية لزيادة التمسك بالدين الإسلامى الحنيف فى كل أقطار الإسلام. ثم إشاعة جو النهضة العلمية فى كل أرجاء المُجتَمع المُسلِم بحيث يتنفس الكل صغاراً و كباراً العلمَ بلغتهم و يتحول تفكيرهم إلى المنهج العلمى الرصين. و غيرها من الأهداف التى تترتب على كل هذا.

و مُحاوَلة تحقيق هذه الأهداف لا يدخل تحت مفهوم (الكلام الكثير) الذى عبَّر به المُعترِض عن شعوره حيال الكلام عن تطوير لغة برمجةٍ عربية. و بصراحةٍ فإن له بعض الحق (بعض و ليس كل) فى إعتراضه هذا؛ بسبب ما رآه من قبل و ما نراه كل يومٍ من مُهاتَراتٍ و شعاراتٍ كاذبةٍ و دعاوٍ براقةٍ ليس لها مضمونٌ حقيقى.

فالإحباط الذى يُسببه كل هذا الزخم يجعل صاحبه غير قادرٍ على تصديق أى شئ يتعلق بذلك الموضوع، و يصير الأمر بالنسبة له مضيعةً للوقت و الجهد، و هكذا يصير الأفضل هو النظر إلى الأمور التى يُمكِن تحقيقها علي أرض الواقع بدون كل أولئك الذين أصابوه بالإحباط و اليأس الشديدين.

و لكنه يُخطِئ حين يُعمِم هذه النظرة على الكل؛ فالحق أنه يجب أن يَقصِرها على المُتحمسين الذين يتحمسون للشئ و نقيضه فى وقتٍ واحدٍ و بنفس الدرجة من الصدق !. من نوعية " صباح الخير يا مان، يلا نعمل لغة برمجة عربى" ثم ينسون الأمر فى لحظات، و إن لم ينسوه فإنهم يُنتِجون شيئاً هزلياً مُضحِكاً ثم يملأون الدنيا حديثا عنه بشكلٍ يُثِير الغثيان فى نفس من هم على شاكِلة صاحب الإعتراض و كذلك كاتب هذه الأسطر.

أما من يتحدث بعقلانيةٍ و منطقيةٍ و يضع لكل أمرٍ حسابه و عدته فيجب أن يُعامَل بطريقةٍ مُختلِفةٍ تماماً، فحينئذٍ يجب الإنضمام إلى حركته تلك و مساعدته بكل الإمكانات المُتاحة لنا. أو على الأقل إن لم يكن بإمكاننا المساعدة بأنفسنا: نُساعده عن طريق الدعم المادى و/أو المعنوى و/أو الدعاية لحركته و الترويج لها بين الشباب القادرين على المُساعَدة.

2- لغة البرمجة نفسها ليست هدفاً و إنما وسيلة لعمل برامج. انظر إلى الدول الأولى عالمياً فى تصدير البرمجيات ستجد أنها لم تعمل لغات برمجة خاصة بها (الهند الأولى عالميا فى صناعة البرمجيات لم نسمع فيها عن لغة برمجة هندية ولم يخرج واحد عبقرى طالب بعمل لغة برمجة هندية). ثم إنه لعمل لغة برمجة محترمة أنت محتاج مجهود سنين وكوادر وتعاون فى العمل بين الأفراد والمؤسسات البحثية وكل هذا والحمد لله ليس موجوداً عندنا. إذاً نعمل بالموجود أم نقول "لأ يا نبدأ من الصفر يابلاش".

صاحب هذا الإعتراض أصاب و أخطأ فى كلامه؛ لقد أصاب حينما قال أن لغة البرمجة وسيلةٌ لا غاية؛ فالحق أن الهدف من إنتاج لغات البرمجة ليس هى بحد ذاتها؛ بل القدرة التى تُتيحها لنا لإنتاج البرامج الأخرى على إختلاف أنواعها و التى تُتِيح لنا التعامل السهل مع البيئة المُحيطة بنا فى المنزل و/أو العمل و/أو شتى مجالات الحياة لتُصبح الحياة أيسر و أفضل.

و كذلك حينما قال أن إنتاج لغة برمجةٍ يحتاج إلى مجهود سنينٍ و كوادرٍ و تعاوُنٍ بين الأفراد و المُؤسَّسات البحثية؛ فإن إنتاج اللغات البرمجية ليس بالأمر السهل كما قد يبدو للمُتحمسين من النوعيات التى تحدثنا عنها قبلاً. و لكى تُنتِج لغة برمجةٍ قويةٍ لها مُميزات لغات البرمجة ذات الشهرة العالمية و التى تُتيح لك أن تبنى عليها و على منتجاتها نهضتك العلمية: فإن الأمر سيكون مشواراً طويلاً من التعلم، و الدراسة، و التصميم، ثم التنقيح، ثم البناء، ثم التحديث، و غيرهن من الأمور التى تبدو سهلة القول إلا أنها تُقاس بالأعمار و الأجيال !.

و لا يُمكِن لفردٍ بمُفرده أن يقوم بكل ذلك إلا إذا كان فى قمة الذكاء و العلم و التفرغ (و هى الصفات التى يستحيل توافرها فى شخصٍ واحدٍ فى زمننا هذا)، و حتى إن كان موجوداً فإن هذا الأمر

سيستحوذ على عمره كله ليصل إلى هدفه. و لكى يتم الوصول إلى الهدف فى زمنٍ معقولٍ و بالإمكانات المعقولة فإن الأمر يستلزم بحقٍ تعاوناً بين الأفراد و المؤسَّسات البحثية للعمل عليه و جعله هدفاً أعلى لها.

هذا ما كان المُعترِض مُحِقاً فيه. أما ما كان مُخطِئاً فيه فهو تلميحه بأنه يجب علينا أن نعمل بالموجود و ألا نبدأ العمل من الصفر؛ و وجه الخطأ هنا هو أننا بالعمل بما هو متاحٌ لنا (أى بِلُغات البرمجة غير العربية و إن كانت مفتوحة المصدر) سنُحقق هدف زيادة المُسَاهمة الإسلامية فى مجال البحث العلمى، و لكننا سنُساهِم فى ترسيخ أقدام اللغة الإنقليزية فى بلادنا أكثر، و بالتالى فى إضعاف اللغة العربية فى بلاد العرب و قتلها فى بلاد المسلمين الأعاجم. و سنكون كالدب الذى قتل صاحبه (في الحكايات الشعبية) حينما أراد إبعاد ذبابةٍ وقفت على وجه الأخير و أزعجته و هو مُستغرقٌ في نومه !.
و الحل كما أفضنا فى الحديث هو: البدء بتعلم العلم الموجود حالياً و إتقانه إلى أقصى الحدود المُمكِنة، ثم استخدام هذا العلم لبدء طريق النهضة، و بالتالى لن نبدأ من الصفر كما يقول صاحب الإعتراض بل باستخدام العلم (و ليس المُنتَج) الذى أنتجه الآخرون، و من ثم نَنقُد و نُصحِّح لنُنتِج نحن بأنفسنا لأنفسنا، و هو الطريق الصحيح للإنتاج العلمى المُستقِل و الشكلِ التعاونى مع الآخرين لأجل فائدة البشرية كلها، و الأكثر تناسباً مع احتياجاتنا في ذات الوقت.

و ثانياً هناك نقطة عدم وجود لغات برمجةٍ غير إنقليزية، و هذه النقطة تدل على أن صاحب الإعتراض لم يبحث الأمر كما يجب حينما قال ما قال، بل أطلق العنان للسانه بدون التحقق من حقيقة ما يقول من عدمها، فهناك لغات برمجةٍ كثيرةٌ تستخدم مبنيةٌ علي لغاتٍ أخري غير اللغة الإنقليزية، بل و علي أبجدياتٍ أخري غير الأبجدية اللاتينية من الأصل !.
و هاكم قائمةٌ لأمثلةٍ على كِلي النوعين منقولةٌ نقلاً من موسوعة ويكيبيديا:

- Aheui – An esoteric programming language similar to Befunge but using Hangul (Korean)
- AMMORIA – Open source object oriented Arabic programming language, designed especially for Arabs.
- Analitik – A Russian-based language for symbolic manipulations with algebraic expressions used in the Soviet series of Mir computers
- ARLOGO – The first open-source Arabic programming language, based on the UCB Logo interpreter

- Chinese BASIC – Chinese-localized BASIC dialects based on Applesoft BASIC; for Taiwanese Apple II clones and the Multitech Microprofessor II
- எழில், Ezhil programming language – A Tamil programming language developed for educational purposes.
- Farsi.NET - A Persian (Farsi, فارسی, پارسی) OO programming language for .NET Framework. It is similar to C# and Delphi.
- Fjölnir – An Icelandic imperative programming language of the 1980s
- FOCAL – Keywords were originally English, but DEC produced versions of FOCAL in several European languages
- 4th Dimension – On local versions, its internal language uses French or German keywords
- Jeem ج – Arabic programming language, based on C++ with simple graphics implementation developed by Dr M A Al-Salka Jeem Website.
- Glagol – A Russian-based programming language similar to Oberon and Pascal
- GOTO++ – A French esoteric programming language loosely based on French and English
- Hindawi Programming System – Indian language set of equivalents for C, C++, lex, yacc, assembly, BASIC, logo, Ada, and others for languages such as Hindi, Gujarati, Assamese, and Bangla (with the BangaBhasha version)
- Hindi Programming Language – A Hindi programming language for the .NET Framework
- hForth – A Forth system with an optional Korean keyword set

- HPL – Hebrew Programming Language
- Kumir - A Russian-based programming language similar to Pascal and IDE, mainly intended for educational usage in schools. The name is an acronym, which means *Комплект ученический 'Мир'* ('Mir' student's environment).
- Lexico – A Spanish OO language for teaching .NET programming
- Linotte – A French programming language
- Logo – In one of its Apple II editions was available in French.
- Loughaty (MyProLang) – A general-purpose natural Arabic programming language based on a proprietary syntax.
- LSE – Langage Symbolique d'Enseignement, a French, pedagogical, programming language designed in the 1970s at the École Supérieure d'Électricité. A kind of BASIC, but with procedures, functions, and local variables, like in Pascal.
- Mama - An educational programming language and development environment, designed to help young students start programming by building 3D animations and games. It is currently available in English, Hebrew, Yiddish, and Chinese.
- MS Word and MS Excel – Their macro languages used to be localized in non-English languages
- ML4 - A language for client/server database programming, with keywords in English or German
- Nadesiko - A Japanese programming language.
- Phoenix - A C-like high-level imperative procedural Arabic programming language

- Rapira – A Russian-based interpreted procedural programming language with strong dynamic type system
- Robik – A simple Russian-based programming language for teaching basics of programming to children
- SAKO – A language created in the 1950s and nicknamed the "Polish FORTRAN"
- Swaram (Tamil) - A simple, general-purpose and procedural language designed for programming in Tamil
- Superlogo – A Dutch creation for computer-aided instruction, based on Logo
- TI-Calculator BASIC – The 68000 version is localized. Unfortunately, various configuration strings are localized too, preventing direct binary compatibility.
- var'aq – A language based on the constructed Klingon language of *Star Trek*
- W-Language – A French programming language used in the WinDev CASE Tool.
- Yiddishe Mama – An educational programming language with a Yiddish version.
- YMB (Yazyk mashin buchgalterskih) – ЯМБ (язык машин бухгалтерских) – A Russian programming language for Iskra-554, Iskra-555, and Neva computers.
- 1C: Enterprise - A Russian framework and language for business applications. English keywords can also be used.

و في هذه القائمة ذُكِر: خمس لغات برمجةٍ عربية، و لغتان هنديتان، و ست لغاتٍ روسية، و ستٌ فرنسية، و لغتان كوريتان، و لغةٌ يابانية، و غيرهن الكثير !

و هذه القائمة لا تُمثِّل كل اللغات غير الإنقليزية؛ فهي قطرةٌ من غيثٍ ليس إلا (كما سأُوضح عند الحديث عن لغات البرمجة العربية فى تالى مُناقَشاتنا بإذن الله تعالي)، و فى هذا الكفاية للرد على الإعتراض بشأن ابتداعنا لمسألة إنتاج لغة برمجةٍ غير إنقليزية. و أظن أننى قد نجحت فى ذلك إلى الحد الذى لا يستطيع معه قارئ الرسالة إلا الرد بنفسه على مثل هذا الإعتراض مُستقبَلًا.

و هذه صورٌ لأسطرٍ برمجيةٍ مكتوبةٍ بلغات برمجةٍ غير إنقليزيةٍ للتمثيل:

لغة Glagol الروسية:

```
ОТДЕЛ Привет+;
ИСПОЛЬЗУЕТ Вывод ИЗ "...\Отделы\Обмен\";
УКАЗ
  Вывод.Цепь("Здравствуй, мир!")
КОН Привет.
```

لغة Lexico الاسبانية:

```
incluya    "System.Windows.Forms"
clase ventana derivada_de "System.Windows.Forms.Form"
{
 publicos:
 mensajes:
   ventana copie "Este es el título de mi primera ventana" en ventana.text
}
```

لغة Rapira الروسية:

```
Проц Старт()
     Вывод 'Hello World !!!'
Кон Проц
```

ترجمة لغة BASIC إلي الصينية:

10 卜=0	10 Y=0
20 入 水, 火	20 INPUT E, F
30 從 日 = 水 到 火	30 FOR A = E TO F
40 卜 = 卜+對數(日)	40 Y = Y + LOG (A)
50 下一 日	50 NEXT A
60 印 卜	60 PRINT Y

ترجمة لغة python إلي الصينية:

```
#!/usr/local/bin/cpython
回答 = 读入('你认为中文程式语言有存在价值吗 ? (有/没有)')

如 回答 == '有':
        写 '好吧, 让我们一起努力!'
不然 回答 == '没有':
        写 '好吧, 中文并没有作为程式语言的价值.'
否则:
        写 '请认真考虑后再回答.'
```

3- مازلنا لم نحل مشاكلنا و معوقاتنا التكنولوجية. فكل ما هنا هو تقليد أكثر منه إبداع. لست ضد التنوع ولكننا لن نفرض أنفسنا كمنافسين بالمزيد من إغلاق أنفسنا بحدو د معلوماتية نضعها و نحن حتى لا ندرى بمعاييرها ولا مقاييسها.

نحن حتى لم نحل مشكلة الخطوط العربية على الإنترنت ولم نصل لحل لوضع معايير خا صة باللغة العربية قد يستخدمها المصنعون العالميون ليتحدثوا إلينا بلغتنا. لا منتجات لدينا لننافس العالم. مازلنا فى المعضلة العربية ألا وهى الناقل و المنقول. و الكم المهول من العقول التى ترفض حتى أخذ العلم بلغته الا صلية.

لست ضد اللغات ولا منشئيها ولكنى أرى أنها يجب ألا تتجاوز المفهوم التعليمى فى هذه المرحلة من تاريخنا التقنى فثقافتنا العربية فى المجال المعلوماتى الآن لا تتجاوز مفهوم الترجمة النصية الغير مكتملة.

هذه اللغات يجب أن تكون و سيلة لحل مشكلتنا أو مشاكلنا الخا صة باللغة العربية قبل أن تستعرض عضلاتها فى عمل برمجيات سطح مكتبية. مرة أخرى لست ضد اللغات و لكن ما الجديد ؟ وما الهدف ؟ ما العائد ؟ و ما الحلول ؟ لا نستطيع المنافسة فى الصعود الى القمر بعربات تجرها الخيول. الأفضل أن نكمل من حيث انتهى الآخرون لا البدء من الصفر وجعل هذا المشروع هو كل غايتنا.

يحتوى هذا الإعتراض على ثلاثة نقاطٍ رئيسةٍ هى:

- وجود مشاكلٍ تقنيةٍ حاليةٍ تُواجِهنا (نحن العرب) لم نحلها بعد، مثل: مُشكِلة الخطوط العربية على الإنترنت، و وضع معاييرٍ خاصةٍ باللغة العربية قد يستخدمها المُصنِّعون العالميون ليتحدثوا إلينا بلغتنا.
- لغات البرمجة العربية يجب ألا تتجاوز المفهوم التعليمى فى هذه المرحلة من تاريخنا التقنى؛ فثقافتنا العربية فى المجال المعلوماتى الآن لا تتجاوز مفهوم الترجمة النصية الغير مُكتمِلة.
- الأفضل أن نُكمِل من حيث انتهى الآخرون لا البدء من الصفر، وجعل هذا المشروع هو كل غايتنا.

أما الاعتراض الثالث فقد رددنا عليه من قبل فلا حاجة لنا إلى تكرار الرد مرةً أخرى، أما الإعتراض الثانى فإجابته بسيطةٌ حقاً؛ فالقول بأن الدور الذى يجب أن تلعبه لغات البرمجة العربية فى المرحلة الحالية هو الدور التعليمى فقط ليس بالأمر المنطقى بتاتاً لسببين اثنين:

- حينما نُصمِّم لغة برمجةٍ عربيةٍ للمجال التعليمى المُمتَد (أي الذي لا يقتصر علي تعليم الأطفال فقط، بل يشمل تعليم بعض الأمور الإحترافية أيضاً لمن هم أكبر سناً) فإن الجهد المبذول فى تصميمها يُمكِن أن يكون قريباً من الجهد المبذول فى تصميم لغات البرمجة التى تُتِيح لنا المُنافسة على المستوى العالمى؛ لأن الفارق بين الإثنين هو الهدف الذى نضعه نُصب أعيننا عند تصميم اللغة الجديدة و بعض الوقت الزائد فى حالة اللغة الأكثر احترافية.

  و الفارق الأكبر بين اللغتين سوف يكون فى المكتبة التى تُبنَى لكلٍ منهما؛ فبينما ستكون فى حالة اللغة التعليمية مكتبةً بسيطةً تكفى فقط لتعلم الأساسيات البرمجية في النقاط المُراد تعلمها فإنها ستكون مكتبةً ضخمةً فى حالة اللغة الإحترافية بحيث يُمكِن استخدامها لبرمجة كافة أنواع التطبيقات (و ليست التعليمية الطابع منها فقط).

و بالتالى فإنه يُمكِننا أن نستثمر هذا الأمر لصالحنا، فنضع من ضمن أهداف اللغة البرمجية التى نُطالِب بها أن تكون سهلة التعلم بحيث نستخدمها منذ بداية خروجها للنور فى العملية التعليمية، قبل أن تُكتَب لها المكتبة الضخمة التى ستتكفل بجعلها قابلةً للتنافس العالمى فى مجال إنتاج التطبيقات البرمجية الإحترافية، و نكون بهذا قد اصطدنا أكثر من عصفورٍ بحجرٍ واحد: فمن الناحية التعليمية ستكون لدينا لغة البرمجة التى تصلح للتعليم من البداية حتي الإحتراف، و على الصعيد الزمنى الأبعد ستكون لدينا لغة البرمجة الإحترافية التى تكفل لنا المنافسة كما نُريد و ذلك دون بذل جهدٍ إضافيٍ لإنتاج اللغة التعليمية، بل دون تعريض من تعلّم لغة البرمجة التعليمية لمشاكل تغيير اللغة

البرمجية من النطاق العربى إلى النطاق الإنقليزى؛ لأنه سيلجأ إلي لغات البرمجة الإنقليزية إن لم يجد لغة البرمجة العربية الإحترافية التى يُكمِل بها مسيرته فيما بعد.

- إن اكتفينا بالدور التعليمى فقط و لم نقم بإنتاج لغة البرمجة العربية الإحترافية فكيف سيتأقلم مَن دَرَس العلم البرمجى بالعربية و بَرْمَج بالعربية على التحصيل العلمى و البرمجة بالإنقليزية ؟!

 و متى سيدرس اللغة الإنقليزية الحديثية التى ستُتِيح له تحصيل العلم أ صلاً ؟!

 و هذا أمرٌ يجب التفكير فيه بروية؛ لأن الأمر لن يكون مع الأجيال القادمة كما هو مع الأجيال الحالية التى تعودت على التعلم بالإنقليزية و ستُواجه بعض الصعوبة فى العودة للتعلم بالعربية. و ستكون هذه الصعوبة قليلة لا لشيءٍ سوى لأنها لغتهم الأم و لأنهم درسوا بها العلوم حينما كانوا أ صغر سناً (علي الأقل هذا حال القسم الأكبر بينهم).

 فالأجيال القادمة التى ستتعلم بالعربية فقط لن يكون التعلم بالإنقليزية مجرد تغيير فى الإ صطلاحات المُستعمَلة بل سيعنى الإنتقال إلى لغةٍ مختلفةٍ تماماً لم يتعودوا عليها عند دراسة العلم بتاتاً، و إلي طرقٍ مُختلفةٍ في التعامل مع المُصطلحات و تكوينها و اختصارها !.

و بالعودة إلى الإعتراض الأول للرد عليه نجد أن المدخل لذلك هو: التساؤل عن السبب وراء عدم وجود المعايير التى تحكم ما يتعلق باللغة العربية. فالحق أن السبب فى ذلك هو عدم وجود مُؤسَّساتٍ عربيةٍ للتوحيدات القياسية تضع تلك المعايير. أو علي الأقل ليست هذه المُؤسَّسات بالقوة التي تُسوِّق بها تلك المعايير و تجعل الآخرين يسيرون عليها بإرادتهم الكاملة.

و ما دام الحال هكذا فمن سيضع تلك المعايير ؟

و إذا وضعها فما الذى سيُعطيها القوة و الإنتشار اللازمين لإقناع الآخرين باستخدامها عند الرغبة فى التحدث إلينا ؟

و إجابة ذلك متعلقةٌ بالنهضة التى نتحدث عنها كل التعلق؛ فحينما ينشأ المُجتمَع التقنى الإسلامى الذى نتحدث عنه و الذى يمتلك أدواته الخاصة و يُنتِج الجديد من العلم فإنه سيكون قادراً على وضع المعايير القياسية. و ستتكفل سمعته و كذا تواجده الإنتاجى الفعلى بدعمها بمنتهى القوة. و سيكونان هما عاملا الإقناع للكل فى الداخل و الخارج. و بالتالى يتم حل تلك المُشكِلات التى تحدث عنها صاحب الإعتراض حلاً جذرياً و بطريقةٍ تُتِيح لنا تطوير حلولٍ لأى مشاكل تُقابِلنا بالإعتماد على أنفسنا و بالتعامل مع الآخر بمُنتهَى النِدِّية.

4- من يريد أن يصنع لغة لا بد من أن يكون قد عمل مبرمجاً لفترة 5 سنين على الأقل. وعنده خبرة باللغات ومواطن ضعفها. و ليس مثل أناس مثلكم (مع احترامى) فيهم شباب لم يعمل. و كل ما يفعلونه

أنهم يسيرون مع الموجة و "نعمل و نسوى". و لا أحد لديه أى qualifications مطلقاً، والمشكلة حتى في أن يطور أحدهم فى لغة موجودة فعلياً؛ و هذا صعب أيضاً لأنك يجب أن تكون أ ساساً مبرمجاً عنده خبرة.

هذا الكلام صحيحٌ تماماً؛ فلكى تقول أنه بإمكانك أن تُصمم لغة برمجةٍ أفضل من الموجود حالياً فيجب أن تكون قد عَمِلتَ فى مجال البرمجة لفترةٍ طويلةٍ (علي الأقل ليست بالقليلة) و بِلُغاتٍ مُختلِفة الأنواع، و أن تكون قد دَرَسْتَ على الأقل العلوم التالية:

- هندسة البرمجيات (علي الأقل الأجزاء التي تحتاجها في الواقع العملي)،
- تصميم لغات البرمجة أو ما يُسمَّي بنظرية لغات البرمجة programming language theory،
- تاريخ تطور لغات البرمجة (علي الأقل تطور اللغات الأ شهر و الأكثر إستخداماً)،

و أن تكون لديك درايةٌ بسيطةٌ بفروعٍ أخري مثل:

- أنظمة التشغيل و واجهات المُبرمِج API المختلفة فيها،
- الشبكات و الأ ساس البرمجى لها،

فإن حَصَّل المُبرمِج هذه العلوم جيداً كان بإمكانه نظرياً نقد لغات البرمجة الموجودة بصورةٍ علميةٍ حقيقية، فإذا ما اكتشف عيوباً و مَساوِئاً فى اللغات الحالية و جَمَع الصفات التى يراها الأفضل فى لغةٍ جديدةٍ أ صبح بإمكانه أن يقول أنه قد صَمَّم لغة برمجةٍ جديدة.

5- سؤال بسيط : انت يا عم المستخدم ضامن ان اللغة دى مثلا مش هيبقى فيها bugs تفتح ثغرات أمنية ؟ لا طبعا لان التطوير مقصور على العرب واحنا بسم الله ما شاء الله بنتعلم أحسن تعليم وعباقرة .... فعلا عباقرة .... هههههه وعدد المشاركين فى تطوير اللغة 2 متحمسين مشتغلوش مثلا فى sun وعارفين الـjava مثلا عملت ايه ولا ايه مزاياها ولا الناس بتستخدم ال C++ ليه.

يحتوى هذا الإعتراض على ثلاثة نقاطٍ رئيسةٍ هى:

- إمكانية وُجود عُيوبٍ برمجيةٍ تفتح الباب أمام الثغرات الأمنية فى بناء لغة البرمجة، و ذلك لعدم وجود الخبرة و التعليم المطلوبين للسيطرة على هذه العملية عند العرب الذين يبنون اللغة.
- عدد المشاركين فى تطوير اللغات سيكون قليلاً.
- لن يكون لدى المُشارِكين فى التطوير أى خبرةٍ فى عيوب و مُميِّزات لغات البرمجة و هو الأمر الذى ينتج عنه ما ذُكِر فى النقطة الأولى.

و مرةً أخري سأرُدُّ على النقطة الثالثة أولًا. فلن نبدأ المشوار الطويل للنهضة إلا إذا ما حَصَّلنا العلوم التى ذَكَرْتُ جزءاً منها من قبل، و لن أسمح لنفسى بأن أنشر الدعوة إلى حركة التطوير العلمى الجديدة هذه قبل أن أُصْبِح ذا خبرةٍ فى مجالى تكفل لى أن أقول فأحسِن القول حتي لا تُصبِح الأمور عبثية. و فى هذا الكفاية.

أما النقطة الاولى فالرد على النقطة الثالثة يُعتبَر رَدّاً عليها أيضاً، و هناك ردٌّ آخرٌ عندى: فالعملية التطويرية لن تكون حِكراً على العرب فقط بل ستكون مفتوحةً أمام المُسلِمين جميعاً، و فى شباب المُسلِمين غير العرب الكثيرون جداً ممن دَرَسوا أفضل الدراسات و تعلَّموا بأفضل الطرق، و فى هؤلاء سَنَدٌ لنا لأنهم يتساوون معنا فى الأحقية فى العمل فى هذا الأمر، فدعوتنا ليست للعرب فقط، بل هى لأجل الإسلام أولًا و آخراً و العرب يستفيدون من ذلك.

أما النقطة الثالثة فأنا أعيها جيداً، و أعلمُ أن المُشارِكين سيكونون فى البداية قليلين للغاية و ربما يقتصر الأمر لسنواتٍ و سنواتٍ علىّ أنا فقط، و لكن هذا الأمر يَهُون إذا ما نَحَّينا العَامِل الزمنى قليلًا، و لا يتأتى هذا إلا إذا ما نذرتُ عمرى كله لهذا الأمر و هو ما قررتُه بالفعل.
ثم إنه أثناء هذا المشوار الطويل و بعد عدة نجاحاتٍ فى مراحل العمل المُختلِفة بإذن الله- عز و جل سيزيد اهتمام الآخرين بالأمر و سيرغبون فى الإنضمام إلىّ فيه، و رُوَيْداً رُوَيْداً سوف تتكون قاعدةٌ من المؤمنين بالقضية يُساعِدون على الوصول للهدف، و ما هى إلا بضع سنين و نكون قد وصلنا إلى جزءٍ كبيرٍ من أهدافنا بفضل الله- تعالي و حينها سنجد الأكثرين و الأكثرين يرغبون فى المُشارَكة؛ لأن النجاح يُوَلِّد النجاح و الفشل (إن استسلمنا له) يُوَلِّد الفشل.

6- غلط إنك تجبر طفل فى صغره وفى طفولته على حاجة معينة؛ انت ايه عرفك أنه لما يكبر هيحب المجال ده. ده اسمه قتل الإبداع. واوعى تقوللى أنه هيستخدم البرمجة فى حاجات مفيدة فى مجاله. بدل كل ده ممكن تعمل لعبة للأطفال توصل لهم قيمة معينة بحاجة كويسة.. و بالمناسبة: المفروض نشجع الأطفال الصغيرين على تعلم الإنقليزية منذ الصغر وإن أمكن لغات أخرى لأن ده بيفتح عقولهم ويخليهم يقدروا لما يكبروا يلاقوا مصادر معلومات تفيدهم وممكن يفيدوا بيها المجتمع بعد كدة. إنما لو خليت كل حاجة عربى يبقى انت بترجع للخلف لأنه مفيش احتكاك بينك وبين الثقافات التانية وانت الأضعف.

هذا الإعتراض يُعتبَر من أسوأ الإعتراضات إن لم يكن أسوأها بالفعل؛ فأسوأ شيءٍ يُمكِنك أن تُعارِض به أمراً ما: هو الشئ الذى يَنسِف حتى ما تُدافِعُ عنه !. فالمُعترِض قال أن تعليم الأطفال البرمجة منذ الصغر يُعتبَر قتلًا للإبداع. ثم عاد و قال أن علينا أن نُعلِّم الأطفال منذ الصغر اللغة الإنقليزية و هذا تضادٌ و تضاربٌ فى كلامه. فلو سلَّمنا جدلًا بصحة كلامه عن خطأ تعليم الأطفال منذ صغرهم علوماً

قد لا يحتاجونها حينما يكبرون: فإنه حينئذٍ يصير من العبث تعليمهم اللغة الإنقليزية لأنهم ربما لا يحتاجونها عند الكِبَر، فلو صار أحدهم نحويا أو عَالِم شَرْعٍ فإنه لن يحتاج إلى تعلم اللغة الإنقليزية و سيُعتَبر تعليمنا إياها له (بمقاييس المُعترِض) قتلًا للإبداع.
و رُوَيْداً رُوَيْداً لن نُعلِّم الأطفال أى شيءٍ لأنهم ربما لن يحتاجوه فيما بعد و سيُعتبَر تعليمنا لهم أى شيءٍ قتلًا للإبداع !.

هذا لو سَلَّمنا بصحة كلامه، و لكن كلامه يُعتبَر من السخافة بمكان؛ فكل الناس تُدرِك أهمية العلوم البرمجية فى حياتنا المُعاصِرة، و الإتجاه الحديث للتعليم فى كل بلاد العالم يسير ناحية التوسع فى تعليم البرمجة و علوم الحاسب للأطفال منذ الصغر، و كذا تطبيق علوم و منهجيات الحَوْسَبة علي كافة مناحي الحياة فيما يُسمَّي بـ(التفكير الحَوْسَبِي computational thinking).
فالعصر الحالى هو عصر الحواسيب و الشبكات، و العصر القادم سيكون لهما أيضاً بكل تأكيد، و ما رأيتُه كطالبٍ فى الهندسة من دخول الحاسوب و برمجياته فى كل المجالات يجعلنى لا أُصدِّق أن أحداً يقول بقول صاحب الإعتراض.

و لكن الأمر يصير واضحاً لو فَكَّرنا أن صاحب الإعتراض لم يقل هذا الكلام عن صحيح اعتقادٍ فيه؛ بل قاله لأنه حاول مُجادَلة أصحاب الدعوى لإنتاج لغة برمجةٍ عربيةٍ و لكنهم حاجُّوه فغلبوه (أو رأي ذلك يحدث لغيره، أو علي الأقل خاف من حدوثه معه)، فلم يَجِد بُدّاً من ترك الأمر الرئيس و ضرب الأهداف التى يقولون أن إنتاج لغة برمجةٍ عربيةٍ سيُحقِّقها، و بالطبع فإن على رأس هذه الأهداف هدف تيسير العلم على أطفال العرب. و لكن لأن الآراء التى يقولها الفرد بدون اقتناعٍ فعليٍ بها تتضارب مع باقى أقواله: فقد تضاربت أقوال المُعترِض مع بعضها البعض كما أوضحنا سلفاً.

أما قول المُعترِض بأن جَعْل كل شيءٍ باللغة العربية يُعَد تخلفاً لأننا سنفقد حينئذٍ الإحتكاك مع الآخر، و بالتالى نفقد القدرة على التعلُّم لأننا الأضعف فهو مُغالَطةٌ واضحة؛ فهذا القول كان سيكون صحيحاً لو أننا اكتفينا بما سنُترجمه فى المرحلة الأولى من المشروع ثم قلنا أننا قد اكتفينا بما أخذناه حتى الآن من الآخرين و لن نأخذ شيئاً آخر، و لكن ما ننويه غير ذلك تماماً: حيث سيظل الإحتكاك بيننا و بين الآخرِ قائماً على أساسٍ من النِدِّية و التعاون المُتبادَل لخدمة الأمة الإسلامية أولاً و التقدم البشرى عامةً ثانياً.

أما تعليم الأطفال لغةً أخرى غير لغتهم الأم فأمرٌ لا نُنكِر أهميته بالفعل، و يكفينا مبدأ **"مَنْ عَـرِفَ لُغَـةَ قَوْمٍ أَمِـنَ شَـرَّهُم"**[3]، و بالفعل فإن من أهم اللغات بالنسبة لنا (كمسلمين و كعرب) اللغتان:

3 هذا ليس بحديثٍ نبوي كما قد يظن البعض، بل هو قولٌ دَارِجٌ يَنسِبه من لا يعلم إلي رسول الله، صلوات الله و سلامه عليه بدون تحقيقٍ أو تدقيق ! فيُرجي التنبه لمثل هذا لأن رسول الله، صلي الله.

الإنقليزية و العبرية، و لو ا ستطعتُ أن أُعلّم كل أطفال المسلمين و شبابهم هاتين اللغتين لفعلت، و لكن ليس بيدى إلا التنبيه و الدعوة.

7- لا أعتقد أن فكرة لغة برمجة عربية هي فكرة جيدة، لأننا لن نستفيد شيئاً من هذه اللغة إلا الكتابة بالرموز العربية وهذا كما أعتقد لسنا بحاجة إليه، لذلك أنا أعتقد أنه يجب تطوير أفكار برمجية جديدة بغض النظر عن اللغة من أجل الرقي بالعرب.

هذا الإعتراض يأخذ مكانه بجانب أخيه السابق له على لائحة أسوأ الإعتراضات؛ فأولاً قال المُعترِض أن فكرة إنتاج لغة برمجةٍ عربيةٍ ليست جيدةً لأننا لن نُحقّق من ورائها إلا فائدة الكتابة بالحروف العربية، ثم قال ثانياً أنه حتى هذا الهدف لسنا بحاجةٍ إليه، ثم ثالثاً أوضح فكرته بكيفية الرُّقى بالعرب عن طريق الأفكار الجيدة بغض النظر عن اللغة.

أما أولًا فقد أفضنا فى شرح الفوائد التى ستعود علينا (نحن المسلمين) من وراء إنتاج لغة برمجةٍ عربية، و لكن الأمر المهم أنها أكثر من مُجرَّد كتابة الأكواد البرمجية با ستخدام اللغة العربية، أما قوله بأننا لا نحتاج إلى الكتابة بالحروف العربية فقولٌ ينبئ عن جهلٍ تام بالواقع و غزوِ الثقافات لبعضها البعض، و يكفينا هنا أيضاً قائمة الفوائد التى تنتج عن البرمجة باللّغة العربية للرد على هذا الغلط.

أما القول الثالث فلا أُنكِر أن نصفه الأول (و هو حاجتنا إلى الأفكار البرمجية الجديدة) صحيح، و لكن العجيب أن المُعترِض و من يحذو حذوه يتنا سون أن بناء لغات البرمجة و أدواتها المُساعِدة يُعَدُّ مجالًا من أهم المجالات البرمجية على الإطلاق، و لا يُنافِسه إلا مجال بناء أنظمة التشغيل، و نهضةٌ برمجيةٌ بدون هذين العِلْمَيْن هى نهضةٌ كَسيحةٌ أقصى طموحٍ لها هو الشحاذة العلمية على أبواب المُنتِجِين الآخرين !.

8- لسنا بحاجة إلى لغة برمجة عربية؛ فلن تجد من يعمل بها سوى الأطفال أو الأشخاص الذين يستصعبون البرمجة بلغات عليا ومعقدة مثل الـC والـjava وغيرها.. لذلك فهؤلاء ليسوا بالأهمية كما المبرمجين العالميين، ولكن لو تم تطوير مبدأ برمجي جديد يغير من المفاهيم الحالية كما ظهرت في السابق البرمجة الشيئية التي قلبت الموازين، فإن ظهور شئ مماثل من القوة هو الأفضل لنا لأنه من المستحيل أن نزحزح أي من اللغات عن عرشها.

عليه و سلم قال: (**إِنَّ كَذِبًا عَلَيَّ لَيْسَ كَكَذِبٍ عَلَى أَحَدٍ. مَنْ كَذَبَ عَلَيَّ مُتَعَمِّدًا فَلْيَتَبَوَّأْ مَقْعَدَهُ مِنَ النَّارِ**) رواه البخاري.

رداً على هذ الإعتراض فإننى أطرح سؤالاً: إذا ما تم إنتاج لغة برمجةٍ عربيةٍ قويةٍ تتفوق على باقى لغات البرمجة فى كثير من النواحى فهل سيكون مُستخدِموها هم فقط من الأطفال و الأشخاص الذين يَسْتَصْعِبُون البرمجة بِلُغاتٍ عُليا و مُعقَّدة مثل الـC و الـjava ، أم أن المُبرمِجين العرب سيستخدمونها بدورهم ؟

فإذا كان الرد بأن المُستخدِمين سيكونون من المُبرمِجين المُحترفِين فإن هذا هو الأمر الذى نطمح إليه، و سيكونون قد أدُّوا واجبهم ناحية أمتهم و دينهم، أما لو كان المُستخدِمون الوحيدون من الأطفال و المبتدئين: فإن الأطفال يكفوننا عند تعليمهم و جعلهم مُبرمِجين مُحترفين فى المُستقبَل لتكملة النهضة العلمية البرمجية، و حينها سيكون الخاسر الوحيد هو أولئك المُتقاعِسين الذين كان بأيديهم المُساعَدة على إعلاء راية الأمة و لكنهم نكسوا رؤوسهم !.

ثم لماذا يهتم المُعترِض بالمُبرمِجين العالميين ؟!

و ما دخلهم بهذا الموضوع من الأصل ؟!

ألسنا نحاول إنتاج لغة برمجةٍ عربيةٍ لتحقيق الإكتفاء الذاتى من العلم و التعليم ؟ فما الداعى للنظر إلى جعل غير المُسلِمين من المُبرمِجين العالميين يُبرمِجون بلغتنا ؟ في حين ربما يُعتبَر هو الأمر المُستحيل مع لغات البرمجة غير العربية؛ لأن المُبرمِج المُحترِف الذى يُتقِن لغته كل الإتقان لا يَميل فى المُعتاد إلى تغيير تلك اللغة إلا فى حالاتٍ استثنائيةٍ و بعد تفكيرٍ طويل؛ و ذلك لأنه يكون قد اعتاد على لغته الإحترافية كل الاعتياد، و تكيَّف مع مُميِّزاتها و عيوبها كل التكيُّف، فأصبَحتْ بالنسبة له منزلًا مُريحاً جميلًا، و من أصعب الأشياء على الإنسان شراء منزلٍ آخرٍ يُكلِّفه مالًا و جهداً إضافيين بدون فائدة، فى حين أنه لديه منزله الفخم.

9- أعتقد أن المشكلة الأساسية والوحيدة أمام إنشاء لغة برمجة عربية هو عدم وجود أو قلة من يتحدثون العربية.

إعتراضٌ آخَرٌ يحتل مكانه فى قائمة أسوأ الإعتراضات؛ فتعداد العرب لا يَقِل عن ثلاثمائة مليون إنسانٍ أغلبهم من فِئتي الشباب و الأطفال، و هذا العدد سيكون هو الهدف الرئيس لأى لغة برمجةٍ عربيةٍ يتم إنتاجها، أما لغة إبداع فلأنها إسلامية الهوية أصلًا فهى مُوجَّهةٌ لأكثر من مليارٍ و نصف المليار من البشر الذين يَستخدِمون العربية فى حياتهم أو على الأقل الجانب الدينى منها، و يَستخدِم كثيرٌ منهم الحروف العربية لكتابة لغاتهم مثل: الفارسية، و الأوردية، و الأمهَرية، و غيرهن من اللغات.

فعلى أقل الإحتمالات تفاؤلًا فإن هناك الملايين يقفون وراء لغة البرمجة العربية نظرياً و يُكوّنون قاعدةً هائلةً من المُرشَّحين لاستخدامها، و بذلك تكون مُشكِلة عدم وجود أو قلة عدد من يتحدثون العربية مُشكِلةً وهميةً لا وُجود لها بتاتاً.

10- لغات البرمجة القوية والتي لا تزال حية، لم تصنع لأن صانعها أراد صنع لغة برمجة فقط لتكون لغة برمجة، بل كل لغة كان لها سبب:

C: صنعت بشكل أساسي لبرمجة نظام UNIX على حد علمي، وانتشرت أيضاً لدعمها الكبير للبرمجة الهيكلية.

C++: صنعت لتسهيل C وإدخال البرمجة غرضية التوجه عليها.

Visual Basic: صنعت لتسهيل البرمجة على نظام ويندوز أو لأغراض تجارية.

وبهذا نرى أن لغات البرمجة المشهورة انتشرت لأنه كان لها أغراض معينة، أو لأنها أدخلت مفاهيم جديدة على البرمجة، مثل لغة Small talk التي أدخلت مفهوم البرمجة غرضية التوجه OOP.

إذاً لتصنع لغة برمجة ناجحة، يجب أن يكون لهذه اللغة مميز ما، ولا تكون مجرد لغة برمجة أخرى.

بدايةً لو كانت اللغة البرمجية العربية التى نطمح إليها لا تُقدِّم فى عالم البرمجة شيئاً إلا إعطاءنا إمكانية البرمجة بمُصطلحاتٍ و حروفٍ عربية فإن هذا أمرٌ رائعٌ جداً، و لا ينتقص من قدرها شيئاً؛ و ذلك لأن هذا يُحقِّق لنا الكثير من الأهداف التى وضعناها أمام أبصارنا حينما فَكَّرنا فى أمر النهضة البرمجية الإسلامية، و لكن الرائع بما لا يُقاس هو أن تُوجَد تلك اللغة البرمجية العربية التى تُقدِّم الجديد إلى العلم البرمجى، و هذا هو ما نود حدوثه و ما سيكون نُصْب أعيننا كما سنُوضِّح فيما يلى.

و من الخطأ مُقارنة لغات البرمجة الأخرى و خروجها إلى الوجود و ظروف ذلك الخروج بلغة البرمجة العربية و ظروف خروجها إلى الوجود؛ و ذلك لأن الغربيين يُمكِنهم الإكتفاء بلغات البرمجة التى لديهم و التى أنتجوها بأنفسهم، و عدم إنتاج لغاتٍ أخرى إلا حين وجود الطفرة البرمجية التى تُغيِّر من مسار علم البرمجة، و لكن الواقع يقول أنهم لم و لن يفعلوا هذا؛ فالواقع يقول أن لغاتٍ أخرى جديدة تخرج إلى الوجود و ذلك لوجود الداعى التجارى لها رغم أنها لا تُحقِّق الطفرة البرمجية الضخمة الملموسة التي تُبرِّر إنتاج لغةٍ كاملةٍ جديدة. و يكون الداعى الوحيد (أو الأكبر) لهذا هو التنافس التجارى بين عمالقة السوق البرمجى فى العالم.

11- طيب عندي لك سؤال: الهنود تعدادهم كبير جداً و متقدمون علينا كثيراً في مجال تقنية المعلومات و معتزون بثقافتهم: فلماذا لا أري نصف لغات البرمجة هندية ؟

الألمان متعصبون ويعتزون بعرقهم ولغتهم: فلم لا أري أكواماً للغات برمجة ألمانية ؟

أنا لا أتعصب ضد اللغة الإنقليزية لأنها أ صبحت لغة قيا سية مثل المعـايير القيا سية. إ ذا ا سـتخدمت لغـات البرمجة المشهورة أكون أ ستخدم نتاج عد د غير محدو د من المستخدمين. طرحي للغة بالإنقليزيـة يعـد رافـد قوي لها. لما ذا نتقوقع تحت مسمى اللغة العربية بدل من الإنطلاق باللغات المتوفرة ؟
أنت تدعوا لمشروع رائد والدافع الوحيد هو التمني تحت مظلة أن العقلية العربية فذة ولا يضاهيها عقلية **!.**

أما القول بعدم وُجو د لغات برمجةٍ بغير اللغة الإنقليزية فقد ر د دنا عليه قبلاً.

أما القول الثانى فهو مُغالطةٌ واضحة؛ فلغات البرمجة منها ما هو مفتوح المصدر و يُشارِك فى تطويره الآلاف حول العالم. و منها ما هو مُغلَق المصدر و لا يملك غير مُصمِّميه قرارات التصميم و مُستقبَل اللغة. و تكون مُشارَكة الآخرين فى تطوير مثل هذا النوع من اللغات عن طريق الإقترا حات التى يُقدِّمونها لمُصمِّمى تلك اللغات لكى يضموها إن ا ستطاعوا إلى اللغة فتزيد من قوتها و كفاءتها. و بالنسبة للغة البرمجة العربية التى نو دها فستكون مفتوحة المصدر يُشرِف على تطويرها فريقٌ خاصٌ يستقبل اقترا حات المُقترِحين و يُدِير النقا شات معهم لتطوير اللغة طوال الوقت. كما يُتابِع مسيرة تطوير باقى اللغات البرمجية ليستفيد منها. و بالتالى لن نخسر من العلم شيئاً بإ ذن الله تعالى بدون الحاجة لجعل لغة البرمجة نفسها باللغة الإنقليزية.

أما القول بأننا نتمنى ما نتمنى تحت مظلة القول بأن العقلية العربية لا تُضاهِيها عقليةٌ أخرى فقولٌ نرفضه قلباً و قالباً في مشروعنا هذا؛ فنحن لا نُفاضِل بين الناس إلا بتقوى الله- عز و جل. و نقول قوله تعالى (**يَا أَيُّهَا النَّاسُ إِنَّا خَلَقْنَاكُمْ مِنْ ذَكَرٍ وَ أُنْثَى وَ جَعَلْنَاكُمْ شُعُوبًا وَ قَبَائِلَ لِتَعَارَفُوا إِنَّ أَكْرَمَكُمْ عِنْدَ اللَّهِ أَتْقَاكُمْ إِنَّ اللَّهَ عَلِيمٌ خَبِيرٌ**) [الحُجُرات:13].
و أي شخصٍ يُريد المُساهمة معنا في هذا المشروع يُمكِنه فعل هذا بدون أي عائقٍ كان.

12- أعتقد أنه من غير المعقول أن نبدأ من الصفر في الوقت الحالي والثورة الكبيرة في التقنية الحديثة. فكل لحظة يتم فيها اكتشاف جديد. بل يجب أن نبدأ من آخر ما و صل له العلم ونحاول أن نطور أنفسنا ومنافسة الغرب في لغتهم وبإ ذن الله نتفوق عليهم.

سَبَق و أن أوضحنا أننا لن نبدأ من الصفر على الإطلاق. و أننا سنبدأ بفهم العلم الحديث و آخر ما و صلت إليه العقول البشرية فى مجالنا. ثم نحاول تطويره أثناء إنتاجنا للغتنا. و بالتالى ستُعتبَر إضافةً إلى العلم البشرى عامة.

13- صراحة ليس لدي الكثير، ولكن بقليل من المنطق ما ذا نستفيد عندما نقوم كل يوم بعمل نفس الفكرة ولكن باللغة العربية ؟. من الأفضل أن يكون هناك لغة برمجة عربية أي من صنع عربي ولكن يتم كتابة أكوا دها بالإنقليزية لأن اللغة الانقليزية هي القيا سية في كل شئ.

في البدء حينما نصنع لغة برمجةٍ عربيةٍ تقوم بعمل نفس الأفكار الموجودة مُسبَقاً فإن هذا علي الأقل يُعَد درساً عملياً عن كيفية تصميم لغات البرمجة و إنتاجها، و هو ما يحدث حول العالم فى الجامعات العريقة عند تدريس مادة المُترجِمات، حيث يقوم الأستاذ بِحَثِّ الطلبة على صُنع مُترجِماتٍ (أو أجزاءٍ من مُترجِماتٍ) للتعلُّم العملى و كسر جُمود الدراسة النظرية، و لكن لأننا هنا لدينا من العباقرة الكثيرون فإنه حتى مثل هذا الأمر يُعَد موضع تساؤلٍ و نقد، بل و استهجانٍ و سخرية !.

ثم إن المُعترِض ناقَض نفسه حينما طالَب بعمل لغة برمجةٍ عربيةٍ باللغة الإنقليزية؛ حيث أن قوله الأول يُطبَّق على قوله الثانى، فلنا نحن أيضاً أن نسأله بقليلٍ من المنطق: ما ذا نستفيد عندما نقوم كل يومٍ بعمل نفس الفكرة و بنفس اللغة ؟!

أما كَوْنُ اللغة الإنقليزية هى اللغة القِيا سِية: فإننا نحن من جعلها لغةً قِيا سِيةً بتخا ذلنا و خنوعنا، و حينما يشتد ساعد الأمة العلمى (بإذن الله تعالي) ستكون اللغة العربية لغةً قيا سيةً بدورها مُعتمِدةً على كميةٍ هائلةٍ من الإنجازات و الإبتكارات التقنية.

14- الوصول الى الأفضل ليست فكرته اللغة، الوصول الى الأفضل فكرته صناعة التميز، و صناعة التميز لا يهم لغتها، اصنع التميز بلغة العالم تصل الى العالم، اصنع التميز بلغتك تصل الى نفسك ومن حولك فقط .

والأهم: أخي الكريم، اللغة العربية من أصعب اللغات، فمثلاً في الإنقليزية (print) هي نفسها للجمع والمؤنث والمذكر، أما العربية فهناك (اطبع واطبعي واطبعوا ولنطبع) إلى آخره، بالنسبة لك أنت تعرفها، لكن ما ذا بالنسبة لشخص يريد تعلم لغة البرمجة هذه ؟.

لهذا أخي الكريم الفكرة المطروحة عن لغة برمجية بالعربية هي فكرة عقيمة، وليست ذات أهميه للمنافسة، أصلاً صناعة لغة برمجة ليست بتلك الأهمية؛ لما ذا تريد أن تصنع لغة برمجة؟ ما ذا تريد أن تضيف على اللغات المصنوعة أيضاً، ما الفكرة الجديدة المميزة التي تمتلكها ولم ينفذوها ؟!

اصنع بلغات البرمجة تلك مشاريع مهمة، facebook لم تصنع لغة برمجة لكنها صنعت موقعاً على الانترنت (و صانعها طالب في سنته الثالثة في جامعة هارفرد) أصبحت قيمة شركته 15 مليار دولار أمريكي، حتى أن مايكروسوفت اشترت بعضاً من أسهمها، ومالكها عمره لم يتجاوز الـ 23 سنة وأصبح رقمها السابعة على العالم وهي في تقدم ..لما ذا لا نصنع هذا ؟!

أُكَرِّر مرةً أخرى أنه لو كنا نطمح إلى الو صول إلى أن نكون الأفضل علمياً فقط لكان هذا القول صحيحاً، و لكننا ذَوو أهدافٍ أخرى لا يُمكِن أن تُحقَّق بهذه الطريقة، و لا يُمكِن تحقيقها إلا بالطريق الذى اخترنا سلوكه و المُضِى فيه حتى آخره.

و لقد ضايقتنى بحقٍ فى هذا الإعتراض مُحاولة المُعترِض تصوير اللغة العربية على أنها لغةٌ صعبةٌ علميا، ضاربا لذلك مثالاً سخيفاً هو أن كلمة print فى الإنقليزية هى نفسها للجمع و المُؤنَّث و المُذكَّر، أما العربية فهناك (اطبع و اطبعي و اطبعوا و لنطبع) إلى آخره، و رداً على هذا أقول للمُعترِض أنه يُمكِننا بمُنتهى البساطة تثبيت صِيغةٍ مُعَينةٍ للأوامر مثل الصِيغة التى ثبتتها لغة (ج) و هى صِيغة (أَفْعَل) مثل (أَكْتُبْ) و (أَرْسُمْ)، أى أن هذا أمرٌ بسيطٌ للغاية.

و إن كان هذا قد ضايقنى كثيراً فإن ما جعلنى أضرب أخماساً فى أسداس قولُ المُعترِض أن إنتاج لغات البرمجة ليس بهذه الأهمية: فحيث أنه لا شئ يُضاف إلى الموجود فعلياً فإن صناعة لغة برمجةٍ أخرى يُعتَبر أمراً عقيماً !، و الحق أننى لن أرد على هذا القول و سأكتفى بالقول بأنه لو أدار مثلُ هذا المُعترِض شركةً برمجيةً لكانت نهايتها؛ ما دام يُفكِّر بطريقة أنه "ليس بالإمكان أفضل مما كان"، علي الرغم من أن هذا المجالٍ لا حد له إلا حدود الإبداع البشرى ذاته.
و عِوضاً عن إضاعة الوقت فى الكلام الفارغ المُسمَّى تصميم لغات البرمجة يقول المُعترِض أن علينا أن نلتفت إلى إنتاج أفكارٍ أخرى مثل الـFacebook، و الذى أصبحت قيمة شركته 15 مليار دولار أمريكي، حتى أن microsoft اشترت بعضاً من أسهمها، ومالكها عمره لم يتجاوز الـ 23 سنة وأصبح رقمها السابعة على العالم !، و إنى لأتساءل عن الإبداع الجديد الذى قَدَّمه الـFacebook مع كل احترامى لفكرته الجميلة ؟!، فليس هناك إبداعٌ خارِقٌ وراءه إلا أنه لبّى رغبة الكثيرين بطريقةٍ أفضل من الموجودة فعلاً، و لكن يظل الأمر أنه لم يُنتِج علماً جديداً يسير بقافلة العلم البشرى إلى الأمام، و أنا أقول هذا رغم نشاطي الشديد علي تلك الشبكة التواصلية، و إدراكي لأهمية وُجودها و الفارق الذي تصنعه في حياة الناس بالسلب أو بالإيجاب.

15- لماذا نضيع وقتاً على إنشاء لغة برمجة عربية ؟! ما الفائدة منها ؟!
إذا علمت الشباب في الجامعات العربية لغة برمجة عربية وتوجهوا إلى سوق العمل، كيف سيبرمجون لمبرمجين أجانب بلغة عربية، لماذا على الأجنبي أن يتعلم العربية وهي ليس إلا العاشرة على ترتيب اللغات العالمية، إضافة إلى أنه (أخي الكريم) ما الهدف أصلاً من لغة برمجة جديدة، ماذا تريد من ميزات لتضعها وهي غير موجودة.

أما التساؤل عن الفائدة المَجْنِيّة من وراء إنتاج لغة برمجةٍ عربيةٍ فأمرٌ تم إيضاحه قبلاً.

أما التساؤل عن كيفية تأقلم الشباب المُبرمِجين بلغة البرمجة العربية مع المُبرمِجين الأجانب فتساؤلٌ فيه مُغالطةٌ واضحة؛ لأن الأجنبى حينما سيُنشِئ فريقاً من المُبرمِجين سيُراعى أن يكون مُختبروا مُنتجَات الفريق و مُصلِحوا عِللِه bugs قادرون على استخدام نفس اللغة البرمجية التى يستخدمها الباقون. و يَصْدُق هذا على الأجانب عامة. فلو أدارت microsoft مشروعاً علمياً يستخدم مُبرمِجوه لغة الـpython مثلاً فيجب أن يكون مُختبِرو التطبيقات المُنتَجة قادرون على البرمجة باستخدام الـpython أيضاً (و لو علي نحوٍ بسيط).

و بالمثل فإنه لو استأجَرَتْ شركةٌ من الشركات البرمجية مُبرمِجين عرباً يُبرمِجون بلغة برمجةٍ عربيةٍ فسيكون عليها جعل مُختَبِري الفريق من اللذين يستطيعون أن يبرُمِجوا بتلك اللغة العربية.
أما إذا اشترطوا استخدام لغةٍ برمجيةٍ غير عربيةٍ (كأن يطلبوا مُبرمِجين يحترفون الـRuby أو الــ Delphi مثلاً) فلن تكون هناك مُشكِلة؛ لأنني أظن أنه يجب على المُبرمِج العربى أن يتعلم (و لو بنسبةٍ صغيرةٍ) الكثير من لغات البرمجة بجانب اللغة البرمجية العربية. و ما قال أحدٌ أن إنتاج لغةٍ برمجيةٍ عربيةٍ يمنع من احتراف لغاتٍ برمجيةٍ غير عربية. على الأقل لتُفِيد المُبرمِج فى مجالات العمل المُختلِفة. و كذلك ليرى بنفسه الفروق التقنية بين لغات البرمجة و الأنواع المُختلِفة منها.
أى أننا سنُعَلِّم الشباب العربى لغة البرمجة العربية ليحترف البرمجة بها. ثم سيكون عليه (بعد الوصول للاحتراف) أن يُحَصِّل بنفسه لغاتٍ أخرى غير عربية. و ستكون المهمة يسيرةً عليه لأنه يُمسِك فعلاً بالأساسيات من تعلمه للغة العربية. و سيكون عليه فقط أن يعرف الفُروق و يضعها فى حُسبانه و هو الأمر الذى جربتُه بنفسى و وجدتُه سهلاً.

أما بالنسبة للتساؤل عن السبب الذي سيدفع المُبرمِج الأجنبى لتعلم اللغة البرمجية العربية فلا محل له من الإعراب في مشروعنا؛ و قد قلنا قبلاً أننا لا نهدف إلي جعل الأعاجِم من غير المُسلِمين يُبرمِجون باللغة العربية. و إنما نستهدف المُسلِمين و العرب.

و أخيراً يتساءل المُعترِض عن الجدوى من إنتاج لغة برمجةٍ جديدةٍ من الأصل. و عن الجديد الذى ستُقدِّمه و لم يتم إنتاجه من قبل على طريقة "ليس فى الإمكان أبدع مما كان". و قد رددنا على ذلك أيضاً فى سابق كلامنا.

16- صراحة هذه مشكلتنا. أننا لا نمد أرجلنا قدر لحافنا و أضغاث أحلامنا كثيرة جداً جداً. ومشروعك أقول لك مستحييييييييل وح تشوف لو في واحد عربي ولا 50 عربي عملوا لغة زي الـ C++ وكلامي مش تنقيص للعرب ولكن كل الظروف عكس التيار.

لو كان مثل هذا القول صحيحاً لَمَا استطاع (لينوس تورفالدز) بدايةً من المرحلة التي كان فيها طالِباً قيادة فريقٍ من مُختلف أنحاء العالم ليُنتِجوا نواة نظام تشغيلٍ يُهدِّد الآن عرش نظام الويندوز windows الذي تُنتِجه microsoft.

و لَمَا استطاعت مجموعةٌ صغيرةٌ من المُبرمِجين المُحترفين أ سّست مشروع الـGNU إنتاج أدواتٍ برمجيةٍ تُعتبَر الآن الأدواتَ المُفضَّلة لكثيرٍ من المُبرمِجين الفَطاحِل و العمادَ الأساسي لأضخم المشاريع مفتوحة المصدر.

بل و لما استطاع الطالب (ماتثياس إترش) تأسيس مشروع واجهة الـKDE حينما لم يُعجِبه حال الواجِهات المرئية لأنظمة التشغيل شبيهة اليُونِكس unix-like، و من ثم صارت من أنجح و أقوي الواجِهات البرمجية علي مُستوَي كافة أنظمة التشغيل الموجودة.

و الحق أن مُشكِلة المُعترِض ليست فى سُوء نظرته إلى العرب، بل إن مُشكِلته الكُبرَى فى تعميمه لهذه النظرة على جميع المُبرمِجين العرب بلا استثناء؛ فمن الحق أن كثيراً من المُبرمِجين العرب و خريجى الجامعات العربية لا يفقهون في العلوم البرمجية إلا بمقدار ما يفهم كاتب هذه الأسطر فى الكيمياء (و أعترفُ أنني لا أفقه فيها شيئاً. سوى أنك لو وَضَعْتَ شيئاً ما مع شئٍ ما ثانٍ لأنتجَ هذا لنا شيئاً ما ثالثاً)، و لكن من الحق أيضاً أن البعض منهم يمتلك قدراتٍ علميةٍ رفيعةٍ تُؤهِّله للمُنافَسة العالمية.

17- سيظل الأساس اللغه الإنقليزية لأن لغة الحاسب هي الأسمبلي، أي أنك تكتب المترجم Complier الذي يترجم أوامر لغة البرمجة سواءاً كانت بلغة عربية أو روسية أو غير ذلك فإنه بالنهايه يترجمها إلى لغة الأسمبلي والتي بالأساس هي لغة الحاسوب، والتي هي بدورها تقوم بمحاكاة الحاسوب ليعمل الأوامر وهي لغة مبنية على الإنقليزية.

ثانياً ليس هنالك منطق أن تقوم بذلك؛ لأنك ستأخذ الوقت الكثير الكثير لبناء لغة أسمبلي ومن ثم لغة برمجة عربية، في حين أن الغرب يتقدم علينا بالتكنولوجيا، يمكننا أن نستخدم ما و صلوا إليه ونحن نبني شيئ جديد فهكذا انتقلت حضارتنا اليهم، أخذوا علومنا وقاموا بالبناء عليها.

فمثلاً أخذوا علوم ابن سينا وأكملوا عليها واكتشفوا اكتشافات جديدة بناءاً على ما قام به ابن سينا في الطب، وغيره الكثير من علماء المسلمين.

هذا الكلام لا يَحوى من العلم إلا مقدار ما يحويه السراب من الماء؛ فالمُترجِم Complier لا يُترجِم الأوامر البرمجية العُليا فى نهاية الأمر للغة التجميع assembly language، بل يُترجِمها إلى لغة التجميع الخاصة بالمُعالِج processor المقصود أولًا، ثم إلى أصفارٍ و آحادٍ فى آخر الأمر.

و لغة التجميع ليست هى لغة الحاسوب الأصلية، بل هى كغيرها من لغات البرمجة الأخرى مُجرّد واجهةٍ لما يفهمه الحاسوب و لذلك فإنها تحتاج بدورها إلى مُترجم من نوعٍ خاص يُسمَّى مُجَمّع assembler يقوم بترجمتها إلى الأصفار و الآحاد، و الإختلاف بينهاً و بين اللغات عالية المُستوَى أن لغات التجميع تكون أوامرها مُناظِرةً تقريباً لأوامر الآلة.

و مثالٌ وهميٌ على ذلك قولنا أن الأمر:

00101010111010100100101010011001

يُكتَب:

ADD R1, R2, R3

فى لغة التجميع الإنقليزية، و يُمكِن أن نصنع برنامجاً يُنتِج نفس الأمر الآلى حينما يقرأ الجملة العربية التالية:

أجمع م1، م2، م3

فهل سيصرخ الحاسوب قائلًا أنه لا يفهم سوى لغة التجميع الإنقليزية، أم أنه لا يهمه إلا الأصفار و الآحاد التى يُمكِن إنتاجها من جُمْلةٍ عربية أو إسبانيةٍ أو هيروغليفية ؟!.

و بالمناسبة: فإن هذه الأرقام التى يستخدمونها هى مُجرّد واجهةٍ أخرى لما يحدث حقيقةً فى عَتَاد الحاسوب؛ حيث أن الصفر ما هو إلا رمزٌ لـ(لا شُحْنَة)، و الواحد رمزٌ لـ(شحنة)، فحتى الأصفار و الآحاد مُجرّد واجهةٍ لما يفهمه العَتَاد الصلب المُكوّن لدارات الحاسوب !
فببساطةٍ حينما نحتاج إلى لغة تجميع نُخاطِب بها الحاسوب عند بناء المُترجم الكامِل: يُمكِننا بمُنتَهى البساطة إنتاج لغة تجميع عربية. و سهولة الأمر فى أن لغة التجميع تكاد لا تكون إلا إنعكاساً جلياً للأوامر الأوّلية المبنيّة داخل المُعالِج.

18- لماذا نحتاج للغة برمجية تستخدم كلمات وعبارات لغة عربية و لغات مثل C و ++C موجودة ؟!
هذا الأمر سيزيد من الفجوة الرقمية بيننا و بين الغرب، في الحقيقة فلنفترض بأننا نستخدم مترجم لغة C يتعامل بالغة العربية، و أردنا برمجة تطبيق به على نظام ويندوز فهذا الأمر سيتطلب تحويل جميع ملفات الترويسة header files و المكتبات إلى مكتبات تستخدم شيفرة مترجم لغة C العربية، عندها سيتطلب هذا الأمر إعادة كتابتها جميعها الخ... هذا بالإضافة للمكتبات المفتوحة المصدر و غيرها مثل OGRE , openGL و.. و.. إلخ.

و الشيء الأكثر أهمية هو أن اللغة العربية يصعب إيجاد اختصارات بها على عكس اللغة الإنقليزية, هذا و نحن لم نذكر بأن مثل هذه اللغات لن تجد من يبرمج بها لأن اللغة العربية يصعب تعلمها و هذا قد يشكل عقبة كبيرة.

القول بأن الفجوة الرقمية بيننا و بين الغرب ستزداد حينما نستخدم لغات برمجةٍ عربيةٍ قولٌ غير صحيح؛ فالفجوة الرقمية تزداد و ستزداد بسبب جهلنا بالتقنيات البرمجية التى يُنتِجها الغرب و يُطوّرها كل يوم. فاللغة وعاءٌ للتقنية المطلوبة و ليست تلك التقنية نفسها. فلو أنتجنا لغات برمجةٍ عربيةٍ تستخدم التقنيات و الأساليب البرمجية الحديثة فلن نتأخر عن الغرب في شئ، و سيكون الأمر كأن تَستخدِم سِماداً من صُنع محليٍ له نفس فائدة السِماد المَصَّنع عالمياً. فالمُهم هنا هو مُكوّنات السِماد و الطريقة و التقنياتُ التى استُخدِمت فى إنتاجه، و ليس مكان صُنعِه أو لغة الكلام المكتوب على الكيس الحاوِى له.

و لو أن لغتنا البرمجية العربية كانت تستخدم الخُوارِزْمات algorithms الأفضل و المنهجيات البرمجية programming paradigms الأحدث و الأقوى لساوَت اللغات الأخرى فى قوتها، و لو استطعنا تطوير خُوارِزْماتٍ أفضل و منهجياتٍ أقوى و طبَّقناها على لغتنا البرمجية لصارت أقوى من باقى اللغات.

و بخصوص المُشكِلة التى يَتوهَّم المُعترِض إمكانية حدوثها عند استخدام مُترجِم لغة C يستخدم اللغة العربية فهى سرابٌ بِقِيعَة؛ لأننا نستطيع إعطاء الماسِح scanner و السَابِر parser القدرة على التعامل مع الكلام الإنقليزى و العربى بنفس الطريقة و ترجمة التعبيرات المُتناظِرة فيهما إلى نفس المُخرَجات، و بالتالي يكون المُترجِم قادراً على ربط الأصفار و الآحاد الناتِجة من ترجمة الكلام العربى و الأصفار و الآحاد الناتِجة من الكلام الإنقليزى (أو العِبرى أو الديموطيقى) ليُنتِج لنا أصفاراً و آحاداً لا يَعلم إلا الله عز وجل مَصدر كلٍ منها.

ثم من قال بأن هذا الهَزْل سيكون هو الهدف الذى نرجوه: إن هدفنا هو إنتاج لغة برمجةٍ عربيةٍ لها مكتبتها الخاصة التى تتفوق على الجميع. و لو احتجنا إلى مكتباتٍ مكتوبةٍ بغير اللغة البرمجية العربية فمن أسهل ما يكون صُنْعُ ما يُشبه الجِسرِ البرمجي لتلك المكتبة يُمَكِّننا من استخدامها. كما يحدث حينما يُريد مُبرمِجو مكتبةٍ مُعيّنةٍ مثل الـOpenCV المكتوبة بلغة الـC إتاحة الفرصة لمُبرمِجى لغةٍ مثل الـpython لاستخدام مكتبتهم. و الفارق الوحيد هو أن جسرنا سيأخذ بعين الإعتبار تَغيُّر اللغة العربية إلى اللغة الإنقليزية فقط.

و هذا الأمر ليس كما يَتوهَّم المُعترِض أمراً جَلَلًا بل هو أمرٌ قريب المَنال. و مثالٌ على ذلك لغة السوبرنوفا (سيلي ذِكْرُها فيما بعد) و التى يُمكِن فيها مزج اللغتين العربية و الإنقليزية فى الكود البرمجى بمُنتَهى الحرية.

19- ليس هذا الذكاء، الذكاء أن تطرح لغة برمجة للعالم، هل تعلم من هم القائمين على لغة الـ php ؟ شركة برمجة اسرائيلية ومقرها "اسرائيل" ولكنها قامت على لغة البرمجة باللغة الإنقليزية.
تخيل فقط أن لغات البرمجة كلٌّ بلغته: ماذا ستدرس في الجامعة، وإذا درّست كل جامعة في كل بلد لغة البرمجة الخاصة بلغة تلك البلد في مايكروسوفت مثلاً عندما يريدون توظيفك وأنت متخرج تعرف تبرمج بالعربية فقط، لماذا سيوظفونك أصلاً، أو مثلاً عندما تقوم بعمل كود وتطرحه كود مفتوح من سيعدل على كودك وهو لا يعرف لغتك، تخيل أن مثلاً أكواد المصادر المفتوحة كانت بالألمانية أو الصينية هل كنت لتستفيد منها، تخيل لو أن شخص جاء اليك وأنت أكبر خبير في البرمجة وقال لك هذا الكود به مشكلة ووجدته باليابانية ماذا ستفعل؟!
اللغة ليست عائق، تعلم البرمجة تعلم لغة بحد ذاتها وعلى العالم كله أن يتعلم لغة واحدة ليتفاهم، الذكاء أن تصنع لغة برمجة في حال كنت في ذاك القدر من الخبرة بالانجليزية تنافس بها العالم

بل الذكاء هو أن تُحدِّد الأهداف التى ترغب فى تحقيقها حق التحديد، ثم تضع لتلك الأهداف طُرُقاً و وسائل يُمكِنك من خلالها أن تصل إليها، و أن تضع تلك الطُرُق و الوسائل فى جدولٍ زمنيٍ جيدٍ يكفل لك ترتيب جهدك و عدم إضاعة طاقتك و يكفيك شر البلبلة و التخبط، و أظن أننا قد فَعَلْنا ذلك حينما قُلنا أن إنتاج لغة برمجةٍ عربيةٍ يُعَد هو حجر الأساس فى أى نهضةٍ برمجيةٍ نرغب فيها، و لا يُمكِننا أن نُكمِل باقى مشوار النهضة إلا إذا ما حقَّقنا هذا الهدف الأول.
و ليس الذكاء هو طرح لغة برمجةٍ للعالم؛ فما الهدف من وراء ذلك ؟، و ما الذى سنستفيده غير رفع إسهامنا العلمى فى المجال البرمجى فقط مُضَحِّين بِلُغتنا و باقى الأهداف التى ببعض الجهد الإضافى سنحصل عليها عند إنتاج لغة برمجةٍ عربية ؟
و المثل الذى ضربه المُعترِض بلغة الـphp لا يَعنينا فى شئ؛ فلو لم تكن هناك لغات برمجةٍ بلغةٍ غير الإنقليزية على الإطلاق لما شَكَّل هذا فارقاً بالنسبة لنا، لأنه من الواجب الدينى علينا أن نُقَوِّى لغتنا العربية و أن نضع الأمة الإسلامية فى مقدمة الأمم حتى لو خَنَعَت باقى الأمم للغرب و أَقَرَّت له بالريادة فى كل مجال.
و لكن الواقع يَرُدُّ هذا الإدعاء و يقول أن هناك لغات برمجةٍ غير إنقليزية و غير عربيةٍ مثلما أفضنا القول من قبل، و من الأقدار أن من ضمنها لغة برمجةٍ بالعبرية و هى لغة HPL – Hebrew Programming Language. فما بالنا نرى الآخرين يقومون بدافعٍ من الحِس القومى أو الحماسة لدينٍ غير صحيح بإنتاج لغات برمجةٍ بلغاتهم و لا نقوم نحن أصحاب دين الله تعالى الخاتم بالدفاع عن لغة ذلك الدين و عن ريادة أمتنا حتى النفس الأخير ؟!

و جزئية سوق العمل قد أجبنا عنها سابقاً، أما جزئية المصادر المفتوحة و العائق اللغوى الذى سينشأ حينما نجعل أكوادنا العربية مفتوحة المصدر فأقول: أنه حقيقةً سوف يمنع حاجزُ اللغة

الغربيين (و خا صةً غير المسلمين بطبيعة الحال) من المُشارَكة معنا فى أكوا د مصا درنا المفتوحة، و لكن أليس فى مليار و نصف مليار مُسلِم من رو سيا و تركيا و الصين و اليابان و فرنسا و جنوب أفريقيا و البرازيل و الهند و أمريكا و مصرٍ و باقى أنحاء الأرض الكفاية ؟!

ألا يَقدِر كل هؤلاء باختلاف ألوانهم و أعراقهم و تعليمهم و بيئاتهم، و ا شتراكهم فقط فى الإخلاصِ لدين الله تعالى و الأخوةِ الرابطة بينهم على جعلنا نستغنى عن أى جهدٍ آخرٍ من الخارج إن ضن به ذلك الخارج علينا ؟

فإن كانوا يستطيعون (و هم يستطيعون و الحمد لله تعالى) فلا مُشكِلة هناك، أما إن لم يكونوا قا درين على ذلك فإن هذه أمةٌ لا تستحق الحياة و من الشرف لها أن تُدفَن تحت الأرض ما دامت لا تقوى بكل هذا العدد من أبنائها بكل الاختلافات و التمايزات التى بينهم على ستر عورتها العلمية بين أمم الأرض !

أما قول المُعترِض أن على العَالَم أن يتعلم لغةً واحدةً ليتفاهم فقد يَقصدُ به أن عليهم تعلم لغةٍ برمجيةٍ واحدة، و هذا أمرٌ يُكَذِّبه الواقع قبل النظر العقلى؛ فما يحدث الآن أن هناك الآلاف من لغات البرمجة باختلاف أنواعها تم إنتاجها فى أزمانٍ مُختلِفةٍ لتُلبّى احتياجاتٍ مُختلِفةٍ فى أماكن مُختلِفة، بل و هناك لكل زمنٍ واحدٍ و كل حاجةٍ واحدةٍ و كل مكانٍ واحدٍ أكثر من لغةٍ تُلبّى طلباته و لا داعى لكثرتها إلا التنافس التجارى بين الشركات، أو الغيرة العلمية بين البشر و مُحاوَلة كلٍ منهم الإدلاء بدلوه فى هذا البحر الخِضّم لعله يُضِيف إلى العلم البشرى و لو أقل القليل.

أما النظر العقلى فيقول بأن التنافس العلمى من الأمور المحمودة؛ لأن الفائز الوحيد فى النهاية حق الفوز هو مقدار العلم الذى وصل إليه البشر، و قدرتهم على تسهيل حياتهم و إنجاز ما يرغبون فيه بأقل جهدٍ مُمكِنٍ بالإعتماد على ما اكتشفَته و ما أنتجَته قرائحهم من علم.

أو قد يَقصِد المُعترِض باللغة الواحدة اللغةَ الإنقليزية ذاتها كلسانٍ مُوحَّدٍ للنشر العلمي، و هذا من الأمور التي فيها مُغالطةٌ واضحة و يُكَذِّبه الواقع أيضاً، فليس لِزَاماً علي العَالَم كله أن يتكلم لغةً واحدةً لكي يتفاهم تقنياً؛ فللترجمة دورها الأساسي إن لم يحدث توحيدٌ للغة البحث العلمي، و لو افترضنا أن اللغة العربية صارت لغةً عالميةً في مجال البرمجة و أصبح من المُعتاد أن تُطرح بها أبحاثٌ علميةٌ برمجيةٌ علي مستويً عالٍ من التميز فإن المُهتمين الغربيين بالتقنية (مثل الشركات الضخمة أو الجامعات الكبري) سوف يُضطرُّون في نهاية الأمر إلي الإهتمام بالترجمة من العربية إلي الإنقليزية.

فإن لم يَهتموا بالترجمة: فسيقع الضرر عليهم و ليس علينا؛ ففي النهاية سوف نحصلُ علي أبحاثهم لأننا نُجِيد لغتهم و سيخسرون هم أبحاثنا لعدم اهتمامهم بما هو خارج دائرة لغتهم و هذا خطأٌ كبير، و قد قلنا قبلًا أننا سنهتم بكل الأبحاث العلمية المُتفرِّدة التي نستطيع الوصول إليها و نُترجمها حتي و إن كانت بِلُغاتٍ أخري غير الإنقليزية.

أما المُنافَسة فهى هدفٌ من أهدافنا حقاً و لكنها تَلِى أهدافاً أخرى من حيث الأهمية كما أوضحنا آنِفا، ثم إن التنافس يكون بالمُنتَج الأفضل تحقيقاً لأهدافه، و لغة البرمجة العربية تُحقِّق لنا أهدافاً أكثر بكثيرٍ من لغات البرمجة غير العربية، و باقى التطبيقات البرمجية التى سنُنافِس بها هى الأكثر أهميةً بالنسبة لغيرِ المسلمين و اللذين ليسوا بعَرَبٍ فى المُنافَسة، و هذه لا يهم فيها أ صَنعتَها بلُغة برمجةٍ عربيةٍ أم بلُغة برمجةٍ إنقليزية؛ فالمهم أن تكون تلك اللغة قادرةً على إنتاج برامج كُفؤةٍ لها سرعة تنفيذٍ عاليةٍ و مقروئيةٍ readability و مكتوبيةٍ writability عاليتين، و كذا إعتماديةٍ كبيرة.

20- تبقى اللغة الإنقليزية أفضل لأن حروفها منفصلة وليس لها أكثر من شكل كما هو الحال في اللغة العربية: أحرف متصلة و أخرى منفصلة، و شكل في أول الكلمة و شكل في منتصف الكلمة و شكل آخر في آخرها، لذلك الترجمة للبرنامج المكتوب باللغة الإنقليزية سيكون أسهل وأكثر سرعة.

هذا كلامٌ لا يقوله إلا شخصٌ:

1) مهزومٌ معنوياً إلى أقصى الحدود المُمكِنة فأصبح لا يرى فيما لديه أى شئٍ حسن، و أصبح يرى ما عند عدوه الذى هزمَه أفضلَ الأشياء و أحسنَها، و تَصْدُق فيه مقولة ابن خلدون **"المَغْلُوبُ مُولَعٌ بِتَقْلِيدِ الغَالِب"**[4] أيما صِدْق؛ ففى نظر هذا التَّعِس فإن اللغة الإنقليزية أفضل فى الإستخدام فى إنتاج لغات البرمجة من اللغة العربية لأن حروفها مُنفصلةٌ و ليست مُتصلة !، ولها شكلٌ واحدٌ و ليست مُتعددة الأشكال ككثيرٍ من حروف اللغة العربية !
و العجب أن هذا هو السبب الوحيد الذى أوْرَده و كأنه أوْرَدَ حُجَّةً من حُجَج ابن حزم ضد مُناظِريه أفحمهم بها فلم يستطيعوا لها رداً، و الحق أنها جعجعةٌ بدون طحنٍ و مُتهافتةٌ قلباً و قالبا.

و يتبين من القول كذلك أن المعترض:

2) لا حَظَّ له من العلوم البرمجية الخاصة بإنتاج المُترجمات أو حتى تصميمها؛ فقد وَقَع فى تقرير أخطاءٍ لا يقع فيها مُبرمِجٌ يفهم تلك الأمور جيداً بحالٍ من الأحوال، فلا دَخْل لاتصال الحروف ببعضها البعض أو انفصالها عن بعضها البعض، و تغير أشكالها أو ثباتها أىُّ دخلٍ فى سهولة و سرعة ترجمة البرامج المكتوبة بلغة البرمجة !.
و الأمور التى تحكم هذه العملية تختلف تماماً عن تلك المُورَدة، و منها:

4 في (المُقَدِّمَة) لابن خلدون.

- طريقة إنتاج مُكوّنات المُترجم المُختلِفة من سَابِرٍ parser و مُحَقّقٍ منطقي semantic analyzer و غيرهما من باقي المُكوّنات، فهل تم إنتاجها با ستخدام أدواتٍ برمجيةٍ مُتخصّصةٍ تأخذ قواعد لغة البرمجة الجديدة المكتوبة بصِيَغ و صفيةٍ قيا سيةٍ مثل الـEBNF و تُخرِج برنامجاً يؤدي دور السَّابِر أو المُحقّق المنطقي للّغة، و ذلك مثل برامج الـyacc و الـANTLR الشهيرة ؟. أم تم إنتاجها بشكلٍ مبا شرٍ يدويا للحصول على أفضل أداءٍ مُمكِن با ستخدام خوارزماتٍ خا صة بالمُبرمِج نفسه ؟.
- خُوارِزْم السَّبْر parsing algorithm المُستخدَم في مُترجم اللغة.
- خُوارِزْمات التحقيق المنطقي semantic analyzing algorithms المُستخدَمة.
- قواعد اللغة البرمجية و مدى سهولة بنائها و التحقق منها برمجياً، و عدم وجود تَضَارُبٍ منطقي فيها.
- شكل الصِيغة التمثيلية الوسيطة intermediate representation في المُترجِم.
- خُوارِزْمات إنتاج الكود الهدف object code.
- خُوارِزْمات تحسين شجرة السَّبْر للبرنامج parsing tree optimization.
- خُوارِزْمات تحسين الكود الوسيط.

و هذه الأمور لا تُؤثّر فقط على سرعة المُترجِم، و إنما تُؤثر كذلك على كل ما يتعلق به من أمورٍ مثل الكفاءة و الإعتما دية و النشاط المُستمِر و غيرها من الصفات الأخرى.

و لكن دعونا نفترض افتراضاً جدلياً أن اللغة الإنقليزية أفضل في بناء لغات البرمجة من اللغة العربية:
فهل سيدفعنا هذا إلى التخلّي عن أمل إنتاج لغة برمجةٍ عربية اللسان ؟
بالطبع لا؛ لأن الأهداف التي نسعى إليها ستجعلنا نُهمِل هذا الأمر تماماً لنُحققها على أرض الواقع، فمثل تلك الإعتراضات حتى و إن صَحّت لن تُقدّم أو تُؤخّر في أمرنا شيئاً و سيظل الحال كما هو، و لكننا ر د دنا عليها فقط لنُزيح الحيرة عن المُحتارِين و لتطمئن قلوب الناس بما نُؤمِن به ليس إلا.

21- لن تجد أب متمرس سوف يجعل ولده يبدأ بلغة عربية بسيطة، بل سيجعله يبدأ من pascal مثلًا، أو حتى php، أو أي لغه أخرى لعدة أسباب:

- اللغه لن يوجد لها مطورين كثر،
- لن يوجد لها دعم كافي،
- لن يجد لها تطوير و إ صلاح للأخطاء،

- لن تجد الكتب،
- لن تجد الأمثلة،
- لن تكون اللغة متقدمة بالشكل الكافي.

يعني في حال تعلم الطفل الأساسيات سوف يترك هذه اللغة، لماذا ؟ لأنها لن تكون مُنتِجة Productive، ناهيك عن صعوبة البرمجة من غير وجود واجهات و خاصة لطفل.

المُعترِض يفترض فى كلامه أن أى لغة برمجةٍ عربيةٍ لا بد و أنها ستكون لها الصفات التالية:

- من الطُرُز البرمجية القديمة و ليست على القدر الكافى من التطور،
- عدد مُطوِّريها قليل،
- ليس لها دعمٌ كبير،
- سيتوقف تطويرها و إصلاح أخطائها بعد فترة،
- ستُعانى من قِلّة الكتب التعليمية و الأمثلة الواقعية التى يتم بها تعليمها للطلاب و الصغار.

و بناءاً على كل ما فات يَستنتِجُ أن هذه اللغة ليس لها مُستقبَلٌ فى تعليم البرمجة للأطفال، و بالتالى فإن الآباء المُتمَرّسِين لن يبدأوا تعليم أطفالهم البرمجة بِلُغةٍ مثلها !، و الحقيقة أنه لو أصبحَ الوضع كما افترض المُعترِض: فبالفعل لن يُخاطِر أى أبٍ بهذا، حتى أنا نفسى إن رزقنى الله تعالى بالولد و أردتُ تعليمه البرمجة فلن أُعلّمه بِلُغةٍ مثل تلك اللغة التى تَحَدّث المُعترِض عن مساويها.

لكننى أتساءل عن السبب الذى سيجعل لغتنا التى نسعى إليها تُعانِى من كل تلك العيوب، فأما العيوب من الثانى و حتى الخامس فهى ستَنْتُج عن تخاذل المُجتَمع البرمجى الإسلامى عن دعم اللغة الجديدة و الوقوف وراءها كلٌّ حسب قُدرَته و فى مجال تخصصه، و بالتالى يكون لى أن أطلب من المُعترِض التخلّى عن موقفه الرافض لأنه سيكون من أكبر أسباب ذلك التخاذل الذى سيُؤدّى إلى وُقوع ما يكرهُ من الأمور !

و أما السبب الأول فلا أعلم ما الذى يدفع المُعترِض لافتراض أن أى لغة برمجةٍ عربيةٍ ستكون قديمةً و غير مُتقدّمة، فالمُفترَض أننا إذا ما حَصّلنا العلم البرمجى و تَشَرّبْناه جيداً كانت لنا القُدرة على استخلاص جوانب القوة و الضعف فى لغات البرمجة العالمية الأقوى حالياً و قديماً، و بالتالى يكون من المُفتَرض أنه بإمكاننا تصميم لغة برمجةٍ تتغلب على الصعوبات الموجودة فى اللغات الحالية و تُضيف الجديد المُفِيد إليها (على الأقل من الناحية النظرية). و هذا هو الناتج الطّبْعِى لمرحلة التعلّم و التّلَقّى من المُنتِجين للعلم قبل مُنافَستهم فى إنتاج الجديد فى هذا العلم.

و مثل هذه النظرة الغير مُنصِفةٍ إلى عقولنا هى من أكبر الأسباب التى تُؤدّى إلى وأد الأفكار الجيدة التى قد تُغيّر دفة الأمور فى عالمنا فى مجالاتٍ كثيرة، و قد تحل من مشاكلنا التى نُواجِهها فى مُجتَمعاتنا الكثير.

و هى نظرةٌ لها خاصية تغذية الذات بأثرٍ رَجْعِى، حيث أنه ما دُمنا ننظر إلى أنفسنا نظرةً دُونِيةً (حتى فى إمكانية المُنافَسة بعد تحصيلنا للعلم المطلوب) فإننا سنُهمِل الأفكار التى كانت ستُفِيدنا فى حل مشاكلنا إن نَفّذناها، و بإهمال هذه الأفكار سيزداد الواقع سُوءاً يوماً بعد يوم. و نتيجةً لتزايد السُوء و تراكمه ستزداد نظرتنا لأنفسنا دونيةً و احتقاراً !

و هكذا ندور فى حلقةٍ مُفْرَغةٍ جهنمية.

22- حتى نجعل أطفالنا مؤهلين لإستعادة مجدنا السابق الذي لم نستطع إستعادته فيجب أن نعلمهم أشياء كثيرة منها تعلم اللغة الإنقليزية من الصغر بإتقان؛ عندما يتعلم اللغة الإنقليزية ويتقنها من الصغر فستفتح أمامه مجالات كثيرة سيبدع فيها، أما إن ضيقنا المجال أمامه بلغة برمجة عربية فلن تفتح له هذه الأفاق الحاسوبية الكبيرة.

إذاً الخلاصة: الغرب لما أخذوا العلم من العرب أتوا للعرب و تعلموا لغتهم و أخذوا علمهم و طوروه و نحن كعرب الأن لسنا مؤهلين لتطوير علم الحاسب عربياً لأننا لم نصل إلى ما وصل إليه الغرب في هذا العلم، فالصحيح أن ننشئ أطفالنا على تعلم اللغة الإنقليزية أفضل بكثير من أن ننشأهم على لغة برمجة عربية.

يَحوِى هذا الإعتراض النقاطَ الرئيسة التالية:

- لا بد من تعليم الأطفال اللغة الإنقليزية حتى تُفتَح لهم أبواب تحصيل العلم البرمجى على مصاريعها؛ نظراً لأن العلم البرمجى الآن يُكتَب مُعظمُ إنتاجه إن لم يكن كلُّه باللغة الإنقليزية.
- تعليم الأطفال لغة برمجةٍ عربيةٍ يُعتبَر تضييقاً لآفاق التعلّم أمام الطفل العربى.
- الصحيح: تَنشِئة الأطفال على لغة برمجةٍ إنقليزيةٍ لا على لغة برمجةٍ عربية، و ذلك حتى نفعل مثلما فعلوا هم حينما كنا الأكثر تقدماً فى العلم و أخذوا هم من علمنا بلغتنا ثم طوروا فيه ليصيروا إلى ما صاروا إليه.

أما النقطة الأولى فلا جَرَم أنها صحيحةٌ كما قلنا في السابق، لكن النقطتين الثانية و الثالثة فيهما من التَّجَنّى الكثير و الكثير؛ لأن إنتاج لغة برمجةٍ عربيةٍ لا يعنى أبداً عدم تعليم الأطفال اللغة الإنقليزية بل و العبرية و الفرنسية و غيرهن أيضا؛ لأن هذه اللغات تَلزَمهم فى باقى أمور حياتهم و ليس بالضرورة الأمور العلمية منها فقط. و قد قلتُ أيضاً من قبل أن تعلُّم اللغات الأخرى ضرورةٌ للأمة. فلا تَعَارض بين الأمرين على الإطلاق.

و تحتوى النقطة الثالثة على مَقولَةٍ صحيحةٍ هى أخذ العلم من مَظَانِّه بلغته، و إيحاءٍ خاطئٍ بأنه يجب على الأطفال منذ الصغر تعلُّم الإنقليزية بالشكل الذى يكفُل لهم تحصيل العلم البرمجى، و هو الأمر الذى لا يُعقَل إذا ما نظرنا إلى ضرورة تدريس الأطفال ما يكفل المُحافَظة على الفِكْر الإسلامى فى عقولهم و عدم تحميلهم ما لا يُطِيقون فى صِغَرهم.

وكذلك فهناك الإيحاء بأن وجود لغة البرمجة العربية فى حد ذاته يَعوق تحصيل العلم من مَظَانِّه، على الرغم من أن وجود اللغة البرمجية العربية التى تُطبِّق النظريات التى تعلمها العرب (من تحصيلهم للعلم البرمجى بالإنقليزية) يُعَد تطبيقاً للأحلام التي تَحمِلها كلمات المُعترِض عن استعادة المجد السابق !

23- أراها مضيعة للوقت؛ فاللغة الإنقليزية هي الأفضل في كل شيء يتعلق بالكمبيوتر وليس فقط بالبرمجة، لست أريد إحباطك: و لكن أظن أن تركز على فهم تقنيات أخرى كتقنية wpf أفضل من إضاعة الوقت على إنشاء لغة برمجة عربية.

و هذا مثالٌ آخرٌ علي الهزيمة النفسية؛ فصاحب الرَّد يُقرِّر أن اللغة الإنقليزية هي الأفضل في كل ما يتعلق بعالم الحَوْسَبة و ليس فقط المجال البرمجي، و مع ذلك لا يقول لنا لِمَ هذا التفوق ؟ و هل هو أصيلٌ في اللغة أم هو نتيجةٌ للتقدم التقني هناك ؟

ثم كانت نصيحته بترك ما "يضيع الوقت" للتفرغ لإجادة التقنيات الجديدة، و هي الدعوة الصريحة للحياة في دور المُستهلِكين إلي أبد الآبدين !

24- أراها مضيعة للوقت و ذلك لأن اللغة العربية ليست موحدة اللهجة و إنما هي بلهجات عديدة و اللغة الإنقليزية موحدة اللهجة فهي أقوى من حيث البرمجة.

سببٌ آخرٌ مُتهافِتٌ للغاية؛ فالبَدَهي أنك حينما تُصمِّم لغة برمجةٍ عربيةٍ فستجعلها بالفُصْحَي، لكن صاحب الرد افترضَ أنها ستكون بلهجةٍ عاميةٍ و هو الأمر الغريب جداً !

أما القول بأن اللغة الإنقليزية مُوحَّدة اللهجة فقولٌ خاطيءٌ تماماً؛ فهناك الإنقليزية البريطانية و الإنقليزية الأمريكية و اللهجة الأسكتلندية و اللهجة الأيرلندية و غيرهن الكثير. و ذلك لاتساع رقعة المُتحدِّثين بالإنقليزية في العالم.

25- أخي أنا مع لغة برمجة عربية لغرض التعليم وليس لغرض البرمجة؛ (لأن اللغات الأساسية مثل لغات ++c، c sharp، vb.net بحر) و لأن المبتدئ وليس لديه لغة إقليزية قوية سوف يمل و يترك البرمجة، في حين إذا كانت لغة البرمجة عربية ربما يستمتع و يتطور و يميل إلى اللغات الاساسية.

و هذا موقفٌ أفضل من سابقيه، لكنه يتصادم مع بقية الأهداف التي نرغب في تحقيقها.

26- أعتقد أنها ستكون بلا فائدة؛ لأن اللغات التي تراها قضت عشرات السنين في التطور كما أن اللغة الإنقليزية أسهل للذين يستخدمون الحاسوب.

لستُ أدري لماذا يَفترِض الجميع أن لغة البرمجة العربية الجديدة ستكون غير حديثة، بينما لا يَفترِضون هذا الإفتراض مع لغات البرمجة الإنقليزية التي تَخرُجُ منها كل حِينٍ واحدةٌ جديدة !، يا أهل الخير: ليس معني أننا سنَصنَعُ لغة برمجةٍ جديدةٍ أنها ستكون مُتخلِّفةً عن المُستوَي الحالي الذي و صلت إليه اللغات الأخري؛ كل ما هنالك هو أننا سنحتاج بعض الوقت لإتمام الأدوات التي سيحتاج لها من سيرغب في البرمجة بتلك اللغة، و علي رأس تلك البرمجيات: المُترجِم compiler و المُفسِّر interpreter و المُنقِّح debugger و غيرهن من الأدوات الأخري التي لا يُستغنَي عنها، و كذلك المكتبة القياسية للُّغة الجديدة.
و مسألة سهولة اللغة الإنقليزية مردودٌ عليها من أكثر من وَجْهٍ فِيما ذُكِر سابِقاً.

نقد لغات البرمجة العربية الحالية

حينما أَخذْتُ أَجمعُ المعلومات عن لغات البرمجة العربية المُتواجِدة بالفعل، أو حتى تلك التى كانت موجودةً ثم اختفت: فُوجِئتُ بكميةٍ كبيرةٍ منها فاقت ما كنتُ أظنه موجوداً بمراحل؛ فقد كنتُ فى بداية الأمر أظن أن مُحاوَلات إنتاج لغة برمجةٍ عربيةٍ لم تبدأ إلا فى السنين الأخيرة، على أساس أن مُعظم البلاد العربية كانت تحت الإحتلال فى العقود الماضية،

ثم بعد التحرر كانت النهضة الإقتصادية هى الأهم بالنسبة للكل و كانت النهضة العلمية تتعلق بمجالات التسليح العسكرى لا أكثر، و لم يكن هناك مكانٌ للحديث عن النهضة العلمية بكامل فروعها إلا فى العصور الحديثة،

و لأن علوم الحاسب كانت أقل العلوم انتشاراً فى بداياتها الأولى عندنا لأنها كانت تتطلب الكثير من الإنفاق، و كانت الأجهزة الحاسوبية الشخصية لم تظهر بعد للنور فكان الشكل الوحيد للحاسوب هو الحاسوب الضخم الذى يحتل مِساحةً كبيرةً و لا يستطيع شراءه إلا المُؤسَّسات الضخمة، و بالتالى لا يملك حق استعماله إلا من قُدِّر له أن يعيش فى بلدٍ يُنفِق على البحث العلمى الكثير، و هو الأمر الذى لم يكن يدور فى خَلَد العرب آنذاك إما بسبب الأنظمة الدكتاتورية الغاشمة التي كانت تحتل مراكز القيادة فيه احتلالاً، و/أو لانشغالهم بقضايا التحرر و هى الأهم و/أو لضعف التعليم أو لغيرها من الأسباب.

المهم أننى كنتُ أظن أن كل ذلك سَيَقِف فى وجه ظُهور مُحاوَلاتٍ لإنتاج لغة برمجةٍ عربيةٍ فيما مضى، و أنه نظراً لاختفاء أو قِلَّة هذه المَوانِع فى هذا العصر فسيكون العرب أكثر إقبالاً على خَوض غِمار هذا الميدان، إلا أننى فُوجِئتُ (و حق لى أن أفاجأ بعد الذي ظننت) أن المُحاوَلات العربية لإنتاج لغة برمجةٍ للعرب قديمةٌ

أكثر مما كنتُ أظن، و أنها بدأت فى العصور التى تلت عصور التحرر من الإحتلال العسكرى الغربي مبا شرة !

و قد فاجأني كذلك العد د الكبير الذى رأيتُه لهذه اللغات؛ فحتى الآن جَمعتُ معلوماتٍ عن أربعين لغة برمجةٍ عربية، و هى اللغات التى توافر لى عنها معلوماتٌ على الشبكة الدولية للمعلومات (الإنترنت)، و ربما كان هناك الأكثر من ذلك و منعني عدم وجو د معلوماتٍ عنها على الشبكة من التعرُّف عليها، لكنني أظن أن هذه اللغات التى جمعتُ عنها معلوماتٍ تكفى لتكوين نظرةٍ شاملةٍ إلى اللغات البرمجية العربية عامةً و عن المسار التطورى الذى اتخذَتْه فى كل مرحلةٍ من مراحل انتشار الوعى بأهمية فرع العلم البرمجى.

وتتفاوت المعلومات التى جمعتُها عن كل لغةٍ من حيث الكم و الكيف عن تلك التى جمعتُها عن الأُخرَيات، و ذلك حسب كمية المعلومات التى وجدتُها على شبكة المعلومات عن تلك اللغة، فهناك لغاتٌ برمجيةٌ كان لها النصيب الأكبر من المَواقِع الشبكية و الكتب التى تشرحها، و ربما كذلك شُروحاتٌ صوتيةٌ و مرئية، فبالتالي كانت لى الفر صة للتعرُّف عليها كل التعرُّف، فى حين أن هناك من اللغات ما لا أعلم عنه إلا ا سمه و جُمْلةً أو جُمْلتَين تَصفانِه با ختصارٍ أو حتى أقل من ذلك !
و بما جمعتُه من معلوماتٍ حر صتُ أول ما حر صتُ على تصنيف تلك اللغات البرمجية العربية إلى أربع مجموعاتٍ رئيسة:

الأولى هى مجموعة اللغات الى لم أحصل على قدرٍ كافٍ من المعلومات فأَسميتُها اللغات المجهولة.

أما **الثانية** فهى فِئة لغات البرمجة التى نقلها المُبرمِجون العرب نقلاً عن لغات برمجةٍ غير عربيةٍ و بالأدق إنقليزية المبني، و لم يكن لهم فضلٌ إلا ترجمة الألفاظ التقنية من اللغة الإنقليزية إلى اللغة العربية، أما الهيكل النحوى الخاص باللغة و كذا كل ما يتعلق بها تقنياً فكان مُطابِقاً للأ صل الإنقليزي تمام المُطابَقة.
كما أنه فى هذا النوع لم يتم إنتاج مُترجِم أو مُفسِرٍ خاصٍ للُّغة، بل كل ما هناك أنه تم إنتاج برنامجٍ يُترجِم البرامج المكتوبة بالصِياغة العربية إلى ملفاتٍ مصدريةٍ مكتوبةٍ باللغة الأجنبية الأ صلية، لذا فهذه اللُّغات لا تُعَدُّ إلا لغات واجهةٍ لا أكثر و ليست لغات برمجةٍ عربيةٍ حقيقية.

و النوع **الثالث** هو اللغات المنقولة: أى التى كانت تسير على هَدْيٍ من اللغات الأجنبية أو حتى تُحاكِيها بصورةٍ تكا د تكون كاملة، لكنها فى ذات الوقت تختلف عن اللغات المُترجَمة فى أنها تمتلك المُترجِم و/أو المُفسِر الخاص بها، و لا تعتمد على المُترجِم و/أو المُفسِر الخاص بِلُغة البرمجة الأجنبية التى تُمثِّل لها الأ صل الذى انحدرت منه.

أما النوع **الرابع** فهو فِئة اللغات الإبداعية، و هى تلك اللغات التى لم تأخذ من أ صلٍ واحدٍ فقط، بل أخذت من أكثر من أ صلٍ و ربما تكون قد نَقدَتْ تلك الأ صول لكى تَبلُغ حداً أفضل منها، مع تطوير الحلول المُناسِبة للعقبات التى تقف أمام اللغة العربية و التى تختلف عن العقبات التى تُواجِه اللغة الإنقليزية عند ا ستخدامهما فى النص البرمجى. و بالطبع فى هذا النوع من اللغات فإن لكل لغةٍ مُترجِمها أو مُفسِرها الخاص بها و الذى بُنِى خصيصاً لأجلها.

و هذا التصنيف لا يأخذ بعين الإعتبار الأجيال البرمجية المُختلِفة للغات البرمجة. فقد نَجد فى فِئة اللغات المنقولة لغاتٍ من الأجيال الحديثة، فِي حين نَجد فى اللغات المُبتَكَرة لغاتٍ من الأجيال الأقدم، فيُرجَى الإنتباه لهذا حتى لا تكون الأمور مُربِكة و يكون ا ستيعاب التقسيم أ سهل و أفضل.

و فيما يلى فإننى أَعرضُ كل قِسْم من هذه الأقسام با ستفاضةٍ لكى أُوضِّح المعلومات الأ سا سية عن كل لغةٍ مع الحفاظ على الإبتعا د عن الصِّياغَات النحوية لها كل البُعْد لأن هذا الأمر لا يهمنا هنا من قريبٍ أو بعيد، و سأوضِّح العيوب و المميزات التى أجدها فِى كل نوعٍ على حِدا، و إن كانت هناك لغةٌ داخل أى فئةٍ تتميز عن باقى لغات فِئتها فسأضع لها نقداً منفرداً خاصاً بها.

إلا أنى كما سيتضح فى تالى الكلام لستُ براضٍ عن تلك المُحاوَلات كل الرضا، أى أننى لم أَجِد من بين تلك اللغات ما يُمكِننى القول بكلِ ثقةٍ أنه يَصلح نقطة بدايةٍ لنا فى مشروعنا، و الأ سباب تختلف من فِئةٍ إلى أخرى و من لغةٍ إلى أخرى، و سأوضِّح فيما يلى الأ سباب الكاملة لذلك فى كل حالةٍ من الحالات.
و سيتضح من هذا الكلام أن هناك لغاتٍ لا تَصلح لأنها ضعيفة البِناء، و أن هناك لغاتٍ أخرى لا تَصلح رغم قوتها و قابليتها للإستخدام بسبب عدم القدرة على تطويرها؛ لإغلاق المُطوِّر لها لمصدرها و رفضه لجعل المصدر مفتوحاً رغم مُحاوَلات المُحاوِلين إثناءه عن ذلك.

قائمةً بلغات البرمجة العربية التى أعلمها:

- اللغات المجهولة:

1) القول
2) المحترف 380
3) الخوارزمى 1978
4) Arabic Programming Language (APL)
5) الحل (أول لغة حا سوب للعرب)
6) لغتى (MyProgLang)
7) دنيا
8) سماء
9) نجلاء (نظام جبري للحا سب الآلي) 1979
10) كاتب CATIB 1984
11) السنبلة 1994
12) Visual Prog 1995
13) Arablan 1995
14) Arab Language 1995
15) عَمُوريا 2006
16) First Arabic Visual Language (FAVL) 2007

- اللغات المُترَجَمة:

1) ليث laith
2) Phoenix
3) فيجوال بازيك العربى
4) خوارزمى 1978
5) غريب 1978
6) سلطانة 1979
7) صخر بيسك 1981-1980
8) ARABW 1986
9) سيناء 1986
10) با سكال العربي 1988

11) ARBI	1990
12) لوقو العربية	1999
13) با سكال المتوازي	2000
14) Arlogo	2006

- اللغات المنقولة:

1) ل.ب.أ (لغة برمجة أخرى)	1986
2) زاى	1998

- اللغات المُبتكَرة:

1) ض	1984
2) لغة برمجة منشيء المواقع SMPL (Sites Maker Programming Language)	
3) خبير	1997-1993
4) الرسالة	2001
5) ج	2001
6) Arabic Assembly Language	2005
7) سوبرنوفا	2009
8) كلمات	2010

اللغات المجهولة

1) لغة كاتِب :

المُطوِّر : د. صبري محمود ، محمد مندورة
سنة البناء: 1984
الجهة الداعمة: جامعة الملك سعود
البلد : السعودية
نظام التشغيل : CPM
مُوا صَفاتٌ أخرى :

ثنائية اللغة، أى أنها تدعم البرمجة بالعربية و الإنقليزية.
بسيطة البناء لتسهيل ا ستخدامها لغير المبرمجين، كالمدرسين مثلاً بغرض تعلمها و ا ستخدامها بكفايةٍ لكتابة و تنقيح الدروس التعليمية بمساعدة الحا سوب.
كانت هناك ورقةٌ علميةٌ تُناقِش تطوير نسخةٍ من اللغة لتعمل على نظام تشغيل الـDOS و لكنى لا أعلم إن تم هذا الأمر أم لا.

2) لغة دُنْيَا :

المُطوِّر : د. زكريا صالح قاسم
سنة البناء: 1978 م
الجهة الداعمة: المؤسسة العامة للمشاريع النفطية بالعراق
البلد : العراق
لغة البرمجة الشبيهة : assembly
مُوا صَفاتٌ أخرى :

و صفٌ نظريٌ للتدريس ولم تُطبَق على أى جهاز.

3) لغة نجلاء :

المُطوِّر : د. رضا سراج الثقة
سنة البناء: 1979 م
الجهة الداعمة: جامعة الملك فهد
البلد : السعودية
لغة البرمجة الشبيهة : لغة البيزك BASIC و لكنها أقوى في الهيكلة
بِنْية المُعالِج : أجهزة الفارابي

مُوا صَفاتٌ أخرى :

تدعم Array و متغيرات Local & Global .

4) السنبلة (Arabic Natural Language Processing):

المُطوِّر : د. الأفندي

سنة البناء: 1994 م

البلد : السعودية

5) Arab Language:

سنة البناء: 1995 م

البلد : البحرين

مُوا صَفاتٌ أخرى :

لطلاب المدارس.

6) Visual Prog:

المُطوِّر : د. خالد سليمان

سنة البناء: 1995 م

البلد : بولدر-أمريكا

مُوا صَفاتٌ أخرى :

رسالة دكتوراه للدكتور خالد سليمان.

7) سماء:

المُطوِّر : خليل الأمين عبد الجواد

بريد المطور: Khal_i_l@yahoo.com

8) المحترف 380:

المُطوِّر : شركة DIGITAL

بِنْية المُعالِج : أجهزة PDP11 و VAX-11

مُوا صَفاتٌ أخرى :

لغةٌ مُعرّبةٌ من الحجم الوَسَط.

اللغات المُترجَمة

1) لغة با سكال المتوازي :

المُطَوِّر : الطالبان: خالد المصيبيح و عبد الله الدكان، تحت إشراف الدكتور عبد الملك السلمان

سنة البناء: 2000 م

الجهة الداعمة: مدينة الملك فهد التقنية السعودية

لغة البرمجة الشبيهة : با سكال

نظام التشغيل : WINDOWS

مُوا صَفاتٌ أخرى :

مشروع تخرج.

2) لغة غريب :

سنة البناء: 1978 م

الجهة الداعمة: جامعة المو صل

البلد : العراق

لغة البرمجة الشبيهة : لغة البيزك

بِنْية المُعالِج : IBM Mainframe

مُوا صَفاتٌ أخرى :

مُوجَهةٌ للمُبتدئين.

لا تستخدم الحروف العربية و تقوم بالعمليات الحسابية فقط.

المُؤشِّر فيها من اليمين إلى اليسار.

تستخدم مترجم بلغة فورتران.

3) لغة ليث LAITH:

مُوا صَفاتٌ أخرى :

شبيهةٌ بلغة الكوبول ومحدو دةٌ جدا.

4) لغة سلطانة :

المُطَوِّر : شركة اوترام

سنة البناء: 1979 م

البلد : السعودية

لغة البرمجة الشبيهة : لغة البيزك

بِنْية المُعالِج : جهاز ZX81.

5) لغة خوارزمي :

سنة البناء: 1979 م

البلد : كاليفورنيا بأمريكا

لغة البرمجة الشبيهة : لغة البيزك لكن بدون تكرار

نظام التشغيل : CP/M

9) لغة صخر بيسك :

المُطَوِّر : شركة صخر

سنة البناء: بين عامي 1980-1981 م

البلد : الكويت

لغة البرمجة الشبيهة : بيزك

مُوا صَفاتٌ أخرى :

بإ ذنٍ من Microsoft الكويت.

10) لغة سينا :

المُطَوِّر : د. الأفندي

سنة البناء: 1986 م

الجهة الداعمة: جامعة الخرطوم

البلد : السو دان

لغة البرمجة الشبيهة : لغة با سكال

مُوا صَفاتٌ أخرى :

ترجمةٌ نظرية.

11) لغة با سكال العربي :

المُطَوِّر : د.حسن مذكور و د.أحمد محجوب

سنة البناء: 1986 م و قيل 1988 م

البلد : السعو دية

لغة البرمجة الشبيهة : با سكال

بنية المعالج : VAX-11

مُوا صَفاتٌ أخرى :

قِيل أنه قد تم الانتهاء من عمل المُترجِم، و قيل أنه قد كُتِب منها الـscanner فقط !.

12) لغة ARABW :

سنة البناء: 1986 م

البلد : البحرين

لغة البرمجة الشبيهة : الكوبول ولكنها مختصرة.

13) ARBI أي Arabic Basic :

سنة البناء: 1990 م

لغة البرمجة الشبيهة : GWB BASIC

نظام التشغيل : DOS

14) لغة با سكال العربي (96) :

المُطَوّر : د.عبدالملك السلمان

سنة البناء: 1996 م

الجهة الداعمة: بمدينة الملك فهد التقنية

البلد : السعودية

لغة البرمجة الشبيهة : با سكال

نظام التشغيل : windows

مُوا صَفاتٌ أخرى :

بيئةٌ برمجيةٌ مُتكامِلة.

15) لغة لوغو العربية :

المُطَوّر : الطالبان: الجهني و الحربي، تحت إشراف د. عبد الملك السلمان

سنة البناء: 1999 م

الجهة الداعمة: مدينة الملك فهد التقنية

البلد : السعودية

نظام التشغيل : WINDOWS

مُوا صَفاتٌ أخرى :

مشروع تخرج.

16) لغة باسكال المتوازي :

المُطوِّر : الطالبان: خالد المصيبيح و عبدالله الدكان، تحت إشراف الدكتور عبد الملك السلمان

سنة البناء: 2000 م

الجهة الداعمة: مدينة الملك فهد التقنية السعودية

لغة البرمجة الشبيهة : باسكال

نظام التشغيل : WINDOWS

مُوا صَفاتٌ أخرى :

مشروع تخرج.

17) arlogo:

سنة البناء: 2006

نظام التشغيل : windows

مُوا صَفاتٌ أخرى :

مشروع تعريب لغة لوقو arlogo.sourceforge.net

نقد اللغات المُتَرجَمة :

1- ليست لغاتٍ بحق، إنما هي مُجرد واجهاتٍ فقط.

2- لها كل عيوب ومثالب اللغات الأصلية.

3- لا تُقدِّم جديداً إلى الفكر الإسلامي باللغة العربية؛ لأنها مجرد إعادة إنتاجٍ للقديم بشكلٍ جديدٍ فقط، ولهذا السبب فلن يمكننا أن نقوم بتطويرها إلى الأفضل لأنها ليست ملكاً لنا أصلاً، كما أن أي تغيير جِدِّيٍ لها ليجعلها تلحق بركب التقنيات البرمجية الحديثة سيقوم بتغيير شكلها، تماماً أي أننا سنقوم بابتكار لغة برمجةٍ جديدة تماماً.

مثلما حدث مع الـVisual Basic .NET التي يعلم كل الناس أنها ليست إلا حيلةً من microsoft حتى لا تخسر عملاءها من مبرمجي VB6، و ذلك كما يقول الأخ: تركي العسيري في كتابه الماتع (Visual Basic للجميع) حيث قال:

ماذا عن Visual Basic.NET؟

يبـدو ان الحـروف NET. تشـد انتبـاه المبـرمجين بعـدما وزعـت Microsoft النسـخ التجريبية Beta مـن جميـع اعضـاء Visual Studio.NET. حسـنا، الاصـدار الاخيـر مـن Visual Basic هو الاصدار السادس VB6 والذي يمثل نهاية الاسـطورة Visual Basic، اما Visual Basic.NET فهي لغة برمجة جديدة لا يكمن الشبه بينها وبين اسـطورتنا إلا الاسـم Visual Basic وصـيغ Syntax بعـض الاوامـر. فالاسـم Visual Basic.NET ليس سوى لعبة تسويقية قامت بهـا Microsoft حتـى لا تخسـر جميـع زبائنهـا مـن مبرمجـي VB1 حتـى VB6. فقبـل ان تشـد الرحـال الـى Visual Basic.NET، فكـر بالموضوع جيدا لان شد الرحال سيكون الانتقال الى لغـة برمجـة جديـدة، كالانتقـال من VB6 الـى C#.

و إذا ما أ صررنا على إيجا د علاقةٍ بين اللغة القديمة (مثل اللوقو) و التطوير الجديد: فـإن الناتـج لن يكون إلا مسخاً لا ينفع فى شيءٍ و لا يُحقِّق أى هدف، أى أن الأفضل هو البدء من الصفر أو كمـا يقول المثل المُعبِّر (على مياهٍ بيضاء).

اللغات المنقولة

1) لغة (ل.ب.أ) أى (لغة برمجة أخرى):

المُطَوِّر : د. فؤاد دهلوي و د. محمد مندورة

سنة البناء: 1986 م

الجهة الداعمة: جامعة الملك عبد العزيز و جامعة الملك سعود

البلد : السعودية

لغة البرمجة الشبيهة : البيزك والبا سكال

مُوا صَفاتٌ أخرى :

كُتِب منها Scanner فقط و لم يكتمل بناء المُترجِم ثم إختفت.

2) لغة زاى:



المُطَوِّر : د. جمال الدين زقور

سنة البناء: 1998 م

الجهة الداعمة: المعهد الوطني للإعلام الآلي

البلد : الجزائر

لغة البرمجة الشبيهة: PASCAL

نظام التشغيل : DOS/WINDOWS

موقع اللغة : http://zegour.uuuq.com/

بريد المبرمج : d_zegour@esi.dz

مُوا صَفاتٌ أخرى :



هى لغة سهلةٌ و بسيطةٌ لكتابة خوارزمياتٍ بسيطةٍ كحساب مجموع عددٍ ما من الأعداد الصحيحة الأولى، و البحث عن الأعداد الأوليّة الأ صغر من عددٍ ما، غيرها.

و للتمرُّن على لوحة المفاتيح بكتابة خوارزمياتٍ عديدةٍ مثل حساب تكرُّر حرفٍ مُعيَّن، أو البحث عن الكلمات التي تبدأ بحرفٍ ما و الكلمات المُتكوّنة من كلماتٍ أخرى، وغيرها.

التمرُّن على آلة الأعداد بكتابة خوارزمياتٍ عديدة، كالبحث عن عددٍ مُعيَّن أو أكبر عددٍ موجودٍ في مُحتوى الآلة، و الأعدادَ التي مُربّعها يُساوي مجموع العدديين السابقين لها في الآلة، وغيرها.

التمرُّن على بعض طرائق خَزْن المُعطَيات البسيطة كالجداول و القوائم الخطية المتسلسلة و البُنـى، و ذلك بكتابة خوارزمياتٍ لجرد مجموعة المُعطَيات المخزونة أو حـذف عنصـرٍ مـا أو زيـادته فـي موضعٍ معيّن.

إمكانية استعمال المُفردات في صِيَغٍ مُناسبةٍ بكتابة الكلمات في صيغة المفرد والجمع (صـحيح: صحاح)، كما يُمكِن اختصار المُفرداًت (اكتب_مباشرة:اك_مب).
و يُمكِن أيضا كتابة خوارزميات الفرز و معالجة عدة جداولٍ و قوائمٍ في الوقت نفسه، بفضـل آلاتها المُجرَّدة. و أيضا كتابة خوارزمياتٍ مُتنوّعةٍ لاستعمال الملفّات أو تصميم تراكيبٍ جديدةٍ لها .

النقد المُوَجّه لهذا النوع من اللغات:

- هى نقلٌ واضحٌ للُّغة الأصل مثل الـPascal، لذا فهى لا تحمل فى طَيّاتهـا إبـداعاً خالصـاً. كمـا أنها نتيجةً لذلك تحمل كل عيوب هذا الأصل و تبتعد عن مميزات غيره، فلغة (زاى) مثلاً:
  - من الأجيال الأقدم و لا تُجارِى التقنيات الحديثة فى البرمجة، مثل البرمجة الكائنية و الوراثـة و التنظيم القوى لملفات البرامج بما يحسن من نوعية التكويد و غيرها، و كـل ذلـك لأنهـا نُقِلـت من لغة الـPascal القديمة.
  - مُغلقة المصدر أى أنها لن تخرج عـن دمـاغ مُطَوّرهـا الـذى لا يجـد الـوقت لتطويرها بمـا فيـه الكفاية، فيظل بالتالى الهدف الوحيد الذى تقـدر علـى أن تُخَصَّـص لأجلـه هـو التعليـم (و هـو الهدف الذى صرّح المُطوّر أنه هو حقاً السبب وراء إنتاج اللغة).
    و لكن حتى فى هذه الحالة سوف يكون مـن الخطـأ تعويـد الأطفـال علـى البرمجـة بالأسـاليب القديمة و بلغةٍ غير الإنقليزية التى سيُضطّرون إلى البرمجة بها بعد ذلك، و سيكون من المُربِك حقاً التحول من العربية إلى الإنقليزية بعد التعود على الأولى.

- هذا النوع كذلك لا يُقدِّم جديداً إلى الفكر الإسلامى باللغة العربية لأنه بـدوره مجـرد إعـادة إنتـاج للقديم بشكلٍ جديـدٍ فقـط، و أى تغييـرٍ جِـدّيٍ لـه ليجعلـه يلحق بركـب التقنيـات البرمجيـة الحديثـة سيقوم بتغيير شكله تماماً، أى أننا سنقوم بابتكار لغة برمجةٍ جديدةٍ بالكلية.
  و إذا ما أصررنا على إيجاد علاقةٍ بين اللغة القديمة (زاى مثلاً) و التطوير الجديد: فإن الناتـج لـن يكون إلا مسخاً لا ينفع فى شيءٍ و لا يُحقّق أى هدف، أى أن الأفضل كما قلنا من قبل هـو البـدء مـن الصفر على مياهٍ بيضاء.

اللغات المُبتكرة

1) لغة ضاد :

المُطَوِّر : د.محمد غزالي خياط بجامعة الملك فهد سابقاً و جامعة الملك عبدالعزيز حالياً

سنة البناء: 1984 م

الجهة الداعمة: مدينة الملك عبد العزيز

البلد : السعودية

لغة البرمجة الشبيهة : BASIC, Pascal, C

نظام التشغيل : DOS و يجري تطويرها على Windows

بِنْية المُعالِج : أولاً على جهاز Cromemco و فيما بعد على الـ IBM-PC

مُوا صَفاتٌ أخرى :

تحتوي هذه اللغة على مُمَيِّزاتٍ جيدةٍ في تراكيب البيانات.

2) لغة خبير:

المُطَوِّر : د.مصطفي عارف و د.حسني المحتسب

سنة البناء: بين 1993 و 1997 م

الجهة الداعمة: جامعة الملك فهد للبترول و المعا دن

البلد : السعودية

نظام التشغيل : CPM

مُوا صَفاتٌ أخرى :

تم تطويرها با ستخدام لغة C لتمتاز بسهولة نقلها من جهازٍ لآخر وبسهولة التكلفة.

3) لغة (ج):

المُطَوِّر : د. محمد عمار السلكا

سنة البناء: 2001 م

نظام التشغيل: WINDOWS

موقع اللغة : www.jeemlang.com

بريد المبرمج: drsalka@jeemlang.com

مُوا صَفاتٌ أخرى :

هى لغة برمجةٍ صُمِّمت خصيصاً من أجل التعليم، و هى فى هذا الجانب تُشبه لغـة (زاى) تماماً و إن كانت تختلف عنها فى أن (ج) هى إبداعٌ صرف، حقيقىٌّ أنه إبداعٌ تنقيحىٌّ إلا أنها لا تزال قريبةً من قِمَّة هرم الإبداع العربى فى مجال ابتكار و إنتاج لغات البرمجة العربيـة، وقـد صَـمَّم المُطَـوِّر برنامجاً لإنتاج و تنقيح تطبيقاتٍ بها هو برنامج (الخوارزمى).

و قد كانت هذه اللغة فى نظرى تصلح نوعاً ما للبدء بهـا لإنتاج لغات برمجةٍ أخرى أقوى إذا ما استمرت عجلة تطويرها باضطراد، و لكن وقفت عدة عواملٍ قويةٍ فى وجه هذا الأمر سأذكرها فى عيوب اللغة فيما يلى.

نقد لغة ج:

1) من الأجيال الأقدم و لا تُجارِى التقنيات الحديثة فى البرمجة، مثل البرمجة الكائنية و الوراثة و التنظيم القوى لملفات البرامج بما يُحسِّن من نوعية التكويد و غيرها.

2) مُغلَقة المصدر أى أنها لن تخرج عن دماغ مُطوِّرها الذى لا يجد الوقت لتطويرها بما فيه الكفاية، فيظل بالتالى الهدف الوحيد الذى تقدر على أن تُخَصَّص لأجله هو التعليم (و هو الهدف الذى صرَّح المُطَوِّر أنه هو حقاً السبب وراء إنتاج اللغة).

و لكن حتى فى هذه الحالة سوف يكون من الخطأ تعويد الأطفال على البرمجة بالأساليب القديمة و بلغةٍ غير الإنقليزية التى سيُضطَّرون إلى البرمجة بها بعد ذلك، و سيكون من المُربِك حقاً التحول من العربية إلى الإنقليزية بعد التعود على الأولى.

3) ليس لها مُترجمٌ إلى لغة الآلة، بل تعتمد على آلةٍ وهميةٍ لتشغيل برامجها.

4) لغة **Arabic Assembly Language:**

المُطَوِّر : د.محمد العدوي، إسماعيل كشك

سنة البناء: 1425 هـ

الجهة الداعمة: جامعة حلوان– كلية الهندسة – قسم الالكترونيات و الاتصالات

البلد : مصر

نظام التشغيل : windows

مُوا صَفاتٌ أخرى :

مطوروا هذه اللغة يُؤكِّدون أنهم قاموا بتطوير لغة أسمبلي العربية وليس تعريباً للغة الأسمبلي الإنقليزية، و يُؤكِّدون أن أوامر هذه اللغة تُترجم مباشرةً للغة الآلة بدون ترجمةٍ وسيطة.

تعمل هذه اللغة على معالجات إنتل 8086، و تَستخدم تنسيق الملفات (.exe) لتوليد الكود الثنائي (Binary Code).



5) لغة سوبرنوفا supernova:

سنة البناء: من يوليو 2009 إلى مارس من العام 2010 م

المُطَوِّر : م. محمود سمير فايد

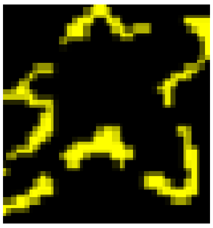
نظام التشغيل: WINDOWS

موقع اللغة : http://supernova.sourceforge.net

بريد المبرمج: msfclipper@yahoo.com

مُوا صَفاتٌ أخرى :

هي لغةٌ من لغات الجيل الخامس، أنتجها المُطَوِّر باستخدام تقنية البرمجة بدون كود (Programming Without Code Technique – PWCT) التي أنتجها هو نفسه أيضاً.



و قد تم إنتاج اللغة في الفترة من يوليو من العام 2009 إلى مارس من العام 2010 (و أظن أن العمل لا يزال مستمراً حتى الآن لتطويرها و صيانتها).

و تهدف هذه اللغة إلى أن تكون أكثر قرباً من اللغة البشرية الطبعية.

و هي السمة العام في لغات ذلك الجيل الحديث. كما أنها لا يبدو عليها أنها تقتصر على الأمور التعليمية فقط، بل يبدو من الواضح أنها لغةٌ خُصِّصت لأجل الإنتاج البرمجي مثل اللغات عالية المستوى المُستَخدمة حالياً مثل Java ، C++ و غيرهن.

و على الرغم من أن هذه اللغة تُعتَبر من اللغات التي تم فيها الإعتماد على الفكر المُجَرَّد دون النقل: إلا أنني أجد عندي انتقاداتٍ حادةً للغاية لها، و ذلك كما سيتم إيضاحه في نقد اللغة بعد قليل.

نقد لغة سوبرنوفا supernova:

- الحشو فيها أكبر من المضمون المهم، لذا فمقروئيتها readability أقل بكثيرٍ من أي لغةٍ أخرى تتجنب هذه الجزئية، كما أن:

- الحشو فيها يُقلّل من مكتوبيتها writeability كثيراً جداً، كما أنه لا يُمكِن استعمال الوسائل الحديثة فى بيئات البرمجة المتكاملة للمُساعَدة على التغلب على هذه الصعوبة لعدم وجود كلماتٍ مفتاحيةٍ فى اللغة يُمكِن للبيئة البرمجية اقتراحها على المبرمج.

و لرؤية الحشو فى اللغة يمكننا أن نرى المثال التالى لبرنامج مكتوب بها:

```
I want window and the window title is Assignment.
The window height is 300. and window back color = (30,200,100).
I want listbox and the listbox name is mylist1.
i want listbox and the listbox name is mylist2.
listbox left is 250.
[v1]. is ( Hello )( World ).
[v2]. = (15)x(3).
[v3]. equal (Country).
[v4]. = (Hello)[v1].
[v5]. = (12) and (7) or (1) not (13).
[v6]. = | (1) [One] (2) [Two] |.
[v7]. = (Hello )[GoodName].
mylist2. listbox selected
listbox must add item "[v1]. is ( Hello )( World )."
listbox must add item "[v2]. = (15)x(3)."
listbox must add item "[v3]. equal (Country)."
listbox must add item "[v4]. = (Hello)[v1]."
listbox must add item "[v5]. = (12) and (7) or (1) not (13)."
listbox must add item "[v6]. = | (1) [One] (2) [Two] |."
listbox must add item "[v7]. = (Hello )[GoodName]."
mylist1. listbox selected
listbox must add from [v1].
listbox must add from [v2].
listbox must add from [v3].
listbox must add from [v4].
listbox must add from [v5].
listbox must add from [v6].
listbox must add from [v7].
```

حيث قد أحطتُ الكلمات ذات المعنى و الحاجة بمُستطيلٍ أحمرٍ لكى أُفرّق بينها و بين الكلمات التى لا حاجة لنا بها و تُشكّل عبئاً فى الكتابة و القراءة.

و عِبؤها فى الكتابة واضح. أما عبء القراءة فيظهر حينما يكون علىّ أن أُطوّر هذا البرنامج فيما بعد؛ فحينما أحاول أن أستخلص المعلومات و الأرقام الهامة من البرنامج فسأواجه فوضىً عارمة و كماً مهولاً من الكلمات أتوه و سطها و لا أخرج بما أريد إلا بعناء.

و حتى لا نكون مجحفين فإن لغاتٍ أخرى مشهورةً لها نفس المشكلة. مثل الـjava التى يشتهر بين نقادها ضرب المثال على الحشو فيها بالبرنامج التالى:

```
public class Hello{
    public static void main(String [] args){
        System.out.println("Hello CCSC");
    }
}
```

و كذلك لغة الـ#C التى يُمكِن ضرب مثالٍ مُشابهٍ لها:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("hello, world");
        }
    }
}
```

و كل هذا لكتابة جُملةٍ بسيطةٍ على الشا شة.

و لكن هذه اللغات لها هدفٌ من وراء هذا الحشو: و هو إجبار المبرمج على ا ستخدام نمط البرمجة الكائنى oop على اعتبار أنه الأ سْلم له و الأفضل تقنياً. و رغم اختلافى مع مصممى اللغتين فى تلك النقطة كل الإختلاف إلا أنه لا يمكننى أن أقارِن حشواً له هدفٌ بحشوٍ ليس له هدفٌ عملي.

- حتى الآن لم يعرف أحدٌ هل ستكون هذه اللغة في النهاية لغةً وظائفيةً أم لغةً كائنية. فكل الشروحات الإلكترونية التى قرأتُها للمُطَوّر (ما كان منها بالعربية و ما كان بالإنقليزية) لا تُقدِّم أى معلومةٍ عن هذه الجزئية.

و لم يتحدث المطور عن دعم: الوراثة. معالجة الإ ستثناءات. الواجهات. و غيرهن من الأمور التى هى عصب لغات البرمجة و عمودها الفقرى.

- يُركِّز المُطَوِّر و يستنفذ جهده و وقته على أ شياءٍ غريبةٍ للغاية فى البناء النَّحْوى للُّغة. مثل:

1- خا صية الكتابة غير المُنظَّمة و الكلمات المُتداخِلة التى تجعل نصاً برمجياً كالتالى:

```
        i
w a n t w i n d o w
a n d
t h e
w i n d o w t i t l
e i s i want window.
```

و كذا النص البرمجى التالى:

```
IwantwindowandtheWindowtitleisHello World.
```

(و هى نفس الأكواد التى ضرب بها المطور المثال على هذه القاعدة) مسموحاً بها فى اللغة !

مع أنه من المعروف أن هذا يُعدُّ عيباً واضحاً فى اللغة إذ أنها هكذا تُساعِد المُبرمِج على كتابة أكوادٍ غير مقروءةٍ بالمرّة !،

و إنى لأتساءل عن نوع المبرمج الذى سيستفيد من هذه الخاصية ؟!.

و فيم سيستفيد بها ؟!

بل إن هذه الخاصية لا تكتفى بخطر إنتاج كودٍ خُرافيٍ غير مقروءٍ فقط، بل تؤثر على أداء مُترجِم اللغة بشكلٍ مُخيف، حيث يقول المطور نفسه:

> اذن كيف يفرق مترجم السوبرنوفا بين الكلمات حتى يصنفها ؟
>
> هذا تحدى كبير اثناء تصميم السوبرنوفا حيث لايكون هناك نهاية للكلمات وانما تتوقع اللغة مع قراءة كل حرف جديد ان هذا الحرف هو نهاية الكلمة ثم تبدا بالبحث عن هذه الكلمة فى النطاقات المسموح بها تبعا لنطاق اخر كلمة تم قراءتها فاذا وجدت الكلمة فانها تعتبر ان هذا الحرف هو نهاية الكلمة اما ان لم تجدها فانها تواصل القراءة باستقبال حرف جديد وتعيد البحث مرة اخرى وهكذا حتى تصل الى الكلمة او الى نهاية البرنامج بدون جدوى من البحث وبالتالى نحصل على رسالة خطأ من المترجم.
>
> وعملية البحث المستمرة فى قاموس اللغة عند قراءة كل حرف تؤدى الى مايكاد ان يكون انهيار فى اداء المترجم والحصول على بطء شديد فى عملية الترجمة ولكن ماينقذ الموقف هو ان الكلمات المتوقعة تكون فى نطاقات محددة مما يقلل فضاء عملية البحث وبالتالى زيادة الاداء نسبيا عما كان سيكون عليه الحال اذا تم البحث فى فضاء اكبر.

فإذا كان الأمر هيناً فى البرامج الصغيرة ذات الأكواد التى لا تزيد عن 1000 سطرٍ فإن الأمر يصير كارثياً فى حالة البرامج الكبيرة، و يصير طامةً كبرى لا حل لها فى حالة إنتاج حزمة برمجياتٍ مترابطة معا يتم بناءها كل مرةٍ فى نفس الوقت.

2- إعطاء المُبرمِج القدرة على تسمية المتغير باسمٍ طويلٍ للغاية من الممكن أن يصل إلى عدة أسطر !.

بل و يُمكن له أن يحتوي علي مسافاتٍ أيضاً !

و هو الأمر الذى لا أدرى كيف سمح به المُطَوِّر ما دام يعلم أنه من الخطأ تقنياً السماح بهذا حيث يقول فى كتابه:

والجدير بالذكر ان الحرية متوفرة فى تحديد اسماء المتغيرات ويمكن ان يشتمل اسم المتغير على مسافات او رموز ويمكن ان يكون الاسم عربى او انجليزى بل يمكن ان يكون اسم المتغير طويل جدا ويمكن ان يكون اكثر من سطر وهى حرية قد تصل الى درجة كبيرة للغاية بمعنى انها حرية زائدة عن الحاجة حيث من غير العملى ان يكون اسم المتغير طويل جدا واكثر من سطر ولكنها امكانية متوفرة ومدعومة من قبل مترجم السوبرنوفا.

فالمعلوم أن واحداً من أهم الأهداف التى يضعها مُصمِّموا لغات البرمجة أثناء تصميم لغة برمجةٍ جديدةٍ هو الأمن، و لا يُقصَد به الحماية من الإختراقات و البرامج الضارة بل يُقصَد به إجبار المبرمج على استخدام طرق برمجةٍ صحيحةٍ تحميه من الوقوع فى الأخطاء البرمجية قدر الإمكان.

و لكن الأمر ها هنا يبدو معكوساً و كأن المطور كان يريد للغته "ميزةً" جديدةً هى عدم الأمان على أساس أنه "على المبرمج أن يقوم بما على المبرمج أن يقوم به".

3- الدَّمج بين اللغتين العربية و الإنقليزية فى النص البرمجى؛ فما دامت سوبرنوفا لغة برمجةٍ عربيةٍ تهدف إلى خدمة العرب و اللغة العربية فى مجال البرمجة فما دخل اللغة الإنقليزية فى هذا الأمر ؟!.

و ما زاد الطين بلة أن المطور يقول فى كتابه:

السوبرنوفا لغة برمجة عربية الهوية من ناحية البناء اما من ناحية الاستخدام فهى تحقق المعادلة الصعبة حيث تتيح ان تكون الشفيرة المصدرية باللغة العربية او باللغة الانجليزية او خليط من اللغة العربية والانجليزية مما يعنى الحفاظ على الهوية العربية وفى نفس الوقت التواصل على النطاق العالمى من خلال اللغة الانجليزية

و إنى لأتساءل عن كيفية المُحافَظة على الهوية العربية إذا كان الكود مكتوباً باللغة الإنجليزية ؟، و أتساءل عن كيفية الحفاظ على التواصل العالمى من خلال اللغة الإنقليزية إذا كان الكود مكتوباً باللغة العربية ؟،

و أتساءل عن كيفية فهم غير العرب للبرنامج المُختلَط بين اللغتين الإنجليزية و العربية ؟.

لذا فإننى أقول أنها ميزةٌ وهميةٌ الغرض الوحيد من وُجودها هو إعطاء اللغة أرضيةً أكبر بين المستخدمين ممن يُبرمِجون بالعربية مع الحفاظ على إمكانية جذب المبرمجين بالإنجليزية، و يحق لى هاهنا أن أنبِه إلى أن برامج الأمثلة التى يُرفِقها المُطوِّر فى كتابه (كتوضيحٍ لما يشرحه

من قواعد) هى فى المقام الأول إنقليزية اللغة، و البرامج العربية تكون تاليةً للبرامج الإنقليزية و كأنها هناك فقط لذر الغبار فى العيون !.

و حتى القواعد اللغوية فأنها فى الأغلب تُذكَر باللغة الإنقليزية ثم يُذكَر لها مثالٌ إنقليزىٌ ثُم يُذكَر المِثال العربى، أى أن القاعدة اللغوية يُذكر نصها بالإنقليزية فقط كثيراً فإذا كنت ممن يريدون ا ستخدام النص العربى فى البرمجة بلغة البرمجة "العربية" سوبرنوفا فيجب عليك أن تستخلص القاعدة من المثال العربى بنفسك. لأنه "يجب على المبرمج أن يقوم بما على المبرمج أن يقوم به".

3- المكتبة الخا صة باللغة ليست إلا واجهةً لمكتبة الـxharbourminiGUI و ليست مكتبةً حقيقيةً مبنيةً باللغة نفسها كما يجدر باللغات القوية أن تكون (بل كما يجدر باللغات العريية الصرفة أن تكون).

حيث أن جانباً قوياً من حوافز إنتاج لغة برمجةٍ عربيةٍ ينتفى عند القيام بأمرٍ كهذا، و هو تعلم كيفية إنشاء المكتبات من الصفر، و بالتالى تَفوتنا الفر صة العملية المتاحة لفهم واحدٍة من أهم أركان النهضة البرمجية لنا.

بل و هناك جانبٌ آخرٌ: أن اللغة البرمجية العربية لن تكون فى هذه الحالة أكثر من واجهةٍ للمكتبة و إمكانياتها، حقيقىٌ أنها ستكون واجهةً مُتقدمةً للغاية عن لغات الترجمة التى أوضحناها من قبل، إلا أنها ستكون لغة واجهةٍ فى آخر الأمر و كل ما تفعله يعو د فى أغلبيته الساحقة فى النهاية إلى إمكانيات المكتبة الحقيقية و إبداع مُطوِّريها و ليس لقوة لغة سوبرنوفا و مُطوِّريها.

- بل إن الأ دهى و الأَمَرَّ أن مُطَوِّر اللغة لم يهتم بتطوير مكتبةٍ قويةٍ لها حتى الآن، و اكتفى ببضعة أوامرٍ بسيطةٍ أخذها من مكتبات DLL خارجية، و هى كما يقول وا صفاً لغته:

> - تشتمل على تعليمات لمعالجة السلاسل الحرفية بمرونة.
> - تشتمل على تعليمات للتعامل مع الوقت والتاريخ.
> - تتيح التعامل مع الملفات بانشائها وقرائتها وكتابتها وتتيح تشغيل البرامج الخارجية.
> - تتيح استدعاء دوال فى مكتبات ربط ديناميكية DLL.
> - تشتمل على مكتبات جاهزة للاستخدام مثل مكتبة تشغيل الصوت ومكتبة تسجيل الصوت ومكتبة تشغيل الفيديو.

و أهمل أ دواتٍ كان يجب توفيرها فى الإ صدارات الأولى من اللغة مثل:

1- **هياكل البيانات الا سا سية** كـ : المُكدِّس stack، الطابور queue، القائمة list، القائمة المرتبة sorted list. و غيرهن.

2- **إجراءاتٍ رياضيةٍ أساسيةٍ** مثل: جيب الزاوية sin، و جيب تمام الزاوية cosine، و ظل الزاوية tan، و غيرهن.

و كان الأجدر به التركيز على هذا الأمر بشكلٍ ضخم على الأقل ليتسنى لنا (نحن المُرشَّحين لنكون مستخدمين للغته) أن نختبرها جيداً، بدلاً من التركيز على الأمور التى اهتم بدعمها فى لغته و أنكرناها عليه فى سابق كلامنا.

7- يضع المُطَوِّر خاصية "عدم الحاجة إلى تعريف المتغيرات" على أساس أنها ميزة، فيقول واصفاً اللغة:

> - تتعامل مع المتغيرات بدون تعريف مسبق وتسند لها قيم ابتدائية بشكل تلقائي حسب سياق الاستخدام.
> - لاتجعلك تهتم بنوع المتغير (حرفي/رقمي...إلخ) وتختار المناسب حسب سياق استخدام المتغير.

و هى فى نظري عيبٌ كبيرٌ من عيوب اللغة عندما تجبر المُبرمِج على جعل كل المتغيرات على هذا النمط؛ إذ أنها تجعل للُّغة خواصاً ضارةً هى:

- عدم القدرة على مُساعدة المُبرمِج على عدم الوقوع فى أخطاء استخدام المتغيرات المختلفة، كأن يستخدم متغيراً يحتوى على قيمةٍ نصيةٍ على أساس احتوائه على قيمةٍ رقمية؛ فلأن اللغة لا تُقيِّد المتغير بنوعٍ مُعيَّنٍ من الأنواع فهذا الخطأ و مثيلاته لن يتم اكتشافها إلا فى زمن التنفيذ، و هو ما يُضِيف إلينا أعباءاً فوق الأعباء فى مرحلة الإختبار لاكتشاف مثل هذا النوعِ من الاخطاء، على الرغم من أن المحاولات الجادة لتطوير عملية اكتشاف الأخطاء تُحاوِل جعل هذا فى مرحلة التصحيح قدر الإمكان.
  و هذا الأمر يُعدُّ الكسر الثانى لقاعدة الأمن فى تصميم لغات البرمجة.

- استهلاك قدرٍ أكبر من الذاكرة و وقت المُعالَجة من ذلك الذى تأخذه اللغات التى تُفرِّق بين أنواع المتغيرات، و هذا أمرٌ مُتوَقَعٌ ما دام المبرمج ألقى جزءاً من مسئوليته على عاتق اللغة لكى تقوم هى به نيابة عنه.
  و قد يكون هذا السلوك معقولاً فى برمجة مواقع الشبكة نظراً لأن تحميل الموقع يأخذ وقتاً أطول من تحميل البرنامج العادى الموجود على القرص الصلب الخاص بالجهاز، و بالتالى لا تُشكِّل الزيادة التى تَحدثنا عنها فارقاً نظراً لتفاهتها مقارنةً بعبء التحميل من الشبكة، و لكنه ليس معقولاً أبداً حينما نتحدث عن برمجة أنظمةٍ حساسةٍ للوقت مثل أنظمة التشغيل و المُترجِمات و المُفسِّرات و غيرهن من الانظمة

المشابهة (و هى الأنظمة الأولى التى يجب أن نهتم بها لإنشاء قاعدةٍ برمجيةٍ قويةٍ تُؤسِّس للنهضة القادمة).

8- ثالِثَة الأَثَافِى [5](كما تقول العرب) أنه يُمكِن لأكثر من كائنٍ واحدٍ أن يحمل نفس الإسم الذى يُستخدَم فى الكود البرمجى !!!

و هو الأمر المُتواتَرُ فى لغات البرمجة رفضُه و عدمُ عقلانيته؛ إذ لو كان لأكثر من كائنٍ نفس المُسمَّى ثم أردنا أن نُجرِى عمليةً ما على أحدها فكيف سيتسنَّى لنا أن نقوم بالتفرقة بينهما ما دام الشئ المُخصَّص للتفرقة بينها (و هو الاسم) فقد هذه الخاصية ؟!

العجيب أن المُطوِّر يقول فى كتابه:

> هل من المنطق ان تحمل اكثر من نافذة نفس الاسم ؟
>
> اعلم ان هذه القاعدة قد يتعجب لها الكثير من المبرمجين لانه في معظم لغات البرمجة الاخرى هذا شيء غير منطقي وغير مسموح به ويسبب خطا لكن هنا في السوبرنوفا مع البرمجة بوصف الخيال الامر يختلف لان الخيال يسمح لنا بذلك وانظر الى النتيجة بنفسك لقد تخيلنا اداء التطبيق ووصفنا هذا الاداء بمرونة واصبح الخيال واقع ولا داعي ان يكون الخيال مقيدا بالاساليب البرمجية المتعارف عليها في لغات لاتقوم على وصف الخيال فالاسلوب مختلف وبالتالي القواعد مختلفة نسبيا.

و كل هذا لأن "الخيال" يسمح بذلك !

و لو كان على القواعد البرمجية أن تتماشى مع "الخيال" حتى و لو كان خطأً لصارت الدنيا خراباً؛ إذ لابد من قواعدٍ صارمةٍ تمنع من الوقوع فى الخطأ و ليس مجاراة الخيال علي عِلّاته.

9- ليس للُّغة حتى الآن خطة عملٍ يُمكِن الركون إليها ممن ينوون التحول إلى البرمجة باستخدامها. فالجدول الزمنى يُعطِى وضوحاً للرؤية و مصداقيةً كبيرةً عند التعامل مع مثل هذه الأمور، كما أنه يُوضِّح التقدم فى تطوير اللغة و هل يسير على الخُطَى المرسومة له أم لا مما يُساعِد على تعديل المسار إذا لزم الأمر.

أما بدون جدولٍ زمني و خطة عملٍ واضحةٍ فإن الأمر يصير بالنسبة للمُطوِّرين المُشاركين تسليةً لا أكثر، و بالنسبة للمُرشَّحين ليكونوا مُستخدِمين للغة البرمجة فإنه يصير حب معرفةٍ لا أكثر. و تتحول

5 و ثَالِثَةُ الأَثَافِي: الْقِطْعَةُ مِنْ الْجَبَلِ يُجْعَلُ إِلَى جَنْبِهَا اثْنَتَانِ، فَتَكُونُ الْقِطْعَةُ مُتَّصِلَةً بِالْجَبَلِ، و ذلك إذا لم يَجِدُوا ثالثةَ الأثافِي. و به فُسِّرَ قولُهم في المَثَلِ: رَمَاهُ اللهُ بِثَالِثَةِ الأَثَافِي: أي بالجَبَلِ: أي بداهِيَةٍ مثلِ الجَبَلِ. قاله ثَعْلَبٌ. (تاج العروس من جواهر القاموس للزبيدي).

العملية كلها إلى عبثٍ لا جدوى منه حتى و إن كـان المُشـارِكون يمتلكون حما ساً كبيراً فى بداية الأمر؛ فالحماس كما جرّبنا جميعنا لا يكفى أبداً و لابد من عقل الأمور بعقالٍ قويٍ مُطَمئِن.

10- اللغة مـن لغـات الجيـل الخـامس، أى أنهـا مـن اللغـات الـتى تهتـم بـأن يكـون الكـو د مكتوبـاً بأ سلوبٍ و تركيبٍ أقرب ما يكون إلى اللغة البشرية العا دية.
و هذا النوع من لغات البرمجة لا يُمكِن الإعتمـا د عليـه إلا فـى مجـالاتٍ مُعينّة مُخصَّـصٍ لهـا، مثـل تطوير أ سـاليب الـذكاء الإ صـطناعى و محاولـة الرقـى بالبرمجيـات الـتى تعمـل علـى إ دارة و تسـيير البشر الآليين لكى يُمكِنها أن تستوعب التعابير و الأوامر البشرية التى تُوجَه لها باللغة المعتا دة حـتى يُمكِن ضم هذه الآلات إلى الخدمة مع البشـر بسـهولةٍ و يُسـرٍ دون الحاجـة إلـى إيجـا د لغـاتٍ ذات قوانينٍ تختلف عن اللغة العا دية للتحكم بها (مثل لغات البرمجة الحالية).
و لأننا (نحن المسلمين عامةً و العرب خا صةً) لم نصل إلى هذه الدرجـة مـن الرُقِـى بعـد: فـإن مـا نحتاجه الآن هو اللغات التى يُمكِننا بها أن نُنافِس على المستوى العالمى فى مجال إنتاج البرمجيات، و يُمكِنها بالتـالى إنتـاج تطبيقـاتٍ تـتراوح بيـن المسـتوى الأ دنـى (أى البرمجـة علـى مسـتوى الخانـة bitwise programming) و المستوى الأعلـى (أى البرمجـة علـى مُسـتوىً عـالٍ مـن التجريد و الوراثة و التنظيم الهيكلى للمكتبات)، و كذلك تُبنَى بها اللغات المنطقية الراقية التكـوين فيما بعد، و هو الأمر الذى لا يتحقق لا فى لغـات الأجيـال الأولـى و الثانيـة و لا فـى لغـات الأجيـال الرابعة و لا الخامسة و إنما تتفر د به لغات الجيل الثالث.

و الغريب أن المُطوّر يقول عن هذا الأمر:

> السوبرنوفا اضافة الى العلم وبداية من حيث انتهى الاخرون وليست اعادة اختراع للعجلة فهي لغة تم بنائها باستخدام اساليب الذكاء الاصطناعي لتكون احدى لغات البرمجة من الجيل الخامس فهي تصنف ضمن فرع معالجة اللغات الطبيعية وتعد اضافة ملموسة الى هذا الفرع من العلم فهي لغة قريبة جدا الى لغة الانسان بنسبة كبيرة تفوق لغات البرمجة الاخرى المتعارف عليها قبل بناء السوبرنوفا.

أى أنه لم يُحاوِل فقط جعلها من لغات الجيل الخامس، بل حاول جعلها تتفوق فـى مسـتوى القُـرب من لغة البشر الطبعية، فإذا ما سلَّمنا للمُطوّر بهذا فإن الإعترا ض يظل موجو داً بأن مثـل هـذه اللغـة خطوةٌ متعجلةٌ على طريق النهضة البرمجية و أن وقتها لم يحن بعد.
أو أنه كان من الخطأ وضعها في مجالٍ ليس مجالها، و كان يجب الإعلان عنها كلغةٍ مُتخصِّصـةٍ غيـر عامة الأغراض حتي يتم تجنب معظم الاعتراضات التي أورتُها هنا.

11- على الرغم من كون بساطة القواعد كان هدفاً أساسياً فى تصميم اللغة فإن هناك قواعداً فى السوبرنوفا تسير على العكس من ذلك، حيث أنها تكون أصعب بكثيرٍ من نظيراتها فى اللغات الأخرى مثل قول المؤلف عن التعبيرات:

- يتم كتابة التعبيرات كالتالي:-
  - يتم وضع اسماء المتغيرات بين اقواس مربعة
    - مثلا [variable name] او [اسم المتغير]
  - يتم وضع القيم بين اقواس
    - مثلا (Value) او (قيمة)
  - يمكن كتابة القيم بشكل لخر للسماح بكتابة الاقواس
    - | Value | او | قيمة |
  - يتم وضع المعاملات Operators المختلفة بين المتغيرات والقيم

و مثالٌ واقعىٌ على القواعد السابقة الجزء التالى من برنامجٍ كتبه المُطوِّر:

```
[x]. = (1). and do while [x] <= (7).
                        listbox must add from [x].
                        Do if [x] == (5).
                                [t]. = .
                                [y]. = (1).
                                Do while [y] <= (5).
                                        [t]. = [t] [y].
                                        listbox must add from [t].
                                        [y]. = [y] + (1).
                                End while
                        End if
                        [x]. = [x] + (1).
                End while
```

حيث نرى أنه قد تَعَسَّف منتهى التَّعسف حينما أرغم المُبرمِج المُستخدِم للسوبرنوفا على إحاطة كل مُتغير بقوسين [ ] و أجبره على إحاطة كل قيمةٍ بقوسين ( ) فى حين أن لغةٍ مثل الـC++ (التى أكرهُها لتعقيدها المقيت) تبدو متسامحةً إلي أقصى الحدود حينما تُقارَن بالسوبرنوفا فى هذا الشأن؛ فلن يحتاج المُبرمِج إلى إحاطة كل رمزٍ من رموزه (التى ربما تبلغ فى المعادلات الرياضية التى تُحَل برمجياً حدها الأقصى) بالأقواس و لا أن يُحيط القيم بالأقواس و لا أن يُراعى الفرق بين أقواس المتغيرات و أقواس القيم.

و يزداد الأمر تعقيداً حينما نأتي إلى التعامل مع المصفوفات حيث نرى ما يشبه الكود التالى:

```
[x]. = (1). and Do while [x] <= (10).
                    listbox must add from [myarray]\[x].
                    [x]. = [x] + (1).
            End while
```

فالمُفترَض فى الكود أن المُطوّر يقوم بأخذ كل العناصر التى فى المصفوفة المُسمّاة (myarray) و يقوم بوضعها فى صندوق قائمةٍ (list box) يُسمَّى (listbox)، و لكن اللافت للنظر هو كيفية استخدامه لعناصر المصفوفة؛ حيث:

- كتب اسم المصفوفة مُحاطاً بقوسين مربعين [ ]
- ثم تلاهما بشرطةٍ مائلةٍ خلفية \
- ليكتب بعدها رقم العنصر المراد استخدامه (بالطبع بين قوسين) مربعين لأن الرقم هنا مُخزَّنٌ فى مُتغير، و لو كان قيمةً مُباشرة لتَحتّم عليه أن يستخدم القوسين () !!!

و لو قارنّا هذا بالـ++C لوجدنا أنها ستكون الأكثر يُسْراً مرةً ثانية؛ فكل ما هنالك أنك ستكتب اسم المصفوفة (بدون أى أقواس) ثم تكتب رقم العنصر بين قوسين [ ] بدون شرطة مائلة خلفية \، و القوسين الوحيدين الذين ستُضطَّر إلى كتابتهما هنا مُوحّدا الشكل و لن يُشكّل فارقاً أن تضع بينهما رقماً مباشراً أو أن تضع مُتغيراً رقمياً.

و الكودين التاليين يُلخِّصان لنا خيبة الأمل الكاملة فى سهولة قواعد سوبرنوفا التى وُعِدنا بها:

حيث فى سوبرنوفا:

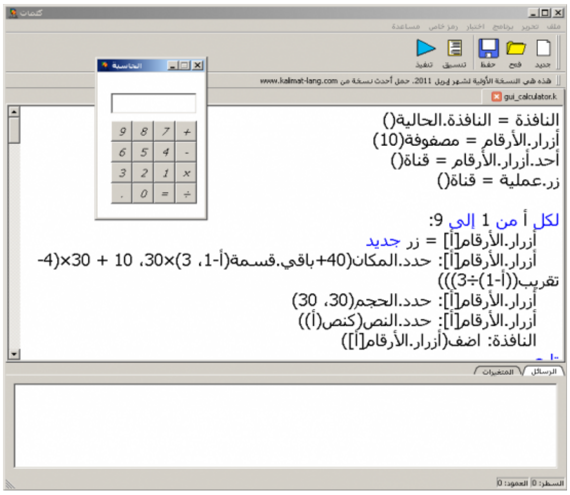
```
[س]. = (1). و نفذ طالما [س] <= (10).
[مصفوفة1]\[س]\(1). = [س].
[مصفوفة1]\[س]\(2). = [س] x [س].
[مصفوفة1]\[س]\(3). = [س] x [س] x [س].
[س]. = [س] + (1).
نهاية طالما
```

يُساوِى فى ++C:

```
x=1;
while(x<=10)
{
    myarray[x][1]  = x ;
    myarray[x][2]  = x * x ;
    myarray[x][3]  = x * x * x;
    x++ ;
}
```

فما بالكم إذا ضربتُ المثل بالـvisual basic أو بالـpython ؟!

الأمر هنا أننا نرى أن الهدف الأصل الذى قال المُطوّر أن لغته تهدف إلى الوصول إليه قد تبخر، و صار بالضبط كسَرَابٍ بقِيعَةٍ نحسبه نحن المخدوعون ماءاً حتى إذا أتيناه لم نجده شيئاً، بل سنجد التّعنُّت الذى لا نُطِيقه.



6) لغة كلمات:

المُطوِّر: م. محمد سامي

مُتأثرة بـ: python و lisp

نظام التشغيل: WINDOWS و GNU/linux

موقع اللغة :

http://www.kalimat-lang.com

بريد المبرمج :

samy2004 @ gmail.com

مُدوَّنة المبرمج:

http://iamsamy.blogspot.com/

مُوا صفاتٌ أُخرى:

يقول م. محمد سامي عن اللغة:

كلمات لغةٌ برمجيةٌ عربيةٌ لتعليم الأطفال البرمجة و قد راعينا فيها سهولة التعلم وقابلية البرامج للقراءة.

اللغة تشمل إمكاناتٍ جاهزةٍ للرسم وتحريك الصور على الشا شة. وبها أيضا إمكانية التعامل مع لوحة المفاتيح والماوس. كما تأتي في بيئة تطويرٍ مُتكامِلة IDE ليسهل بها كتابة وتنفيذ البرمجيات.

ربما تكون كلمات لغة للأطفال لكنها ليست لعبة. هذه لغةٌ حقيقيةٌ بأو صافٍ مألوفةٍ للمُبرمِجين المُحترفين:

1. Object-oriented programming
2. Dynamic typing
3. Garbage collection
4. عمل ملفات تنفيذية exe (انظر قائمة برنامج-> عمل ملف تنفيذي)

ولعلّها تكون بإذن الله جزءاً من نهضةٍ علميةٍ في أمتنا.

ميزات لغة كلمات:

بدايةً: تُعد لغة كلمات أرقي لغات البرمجة العربية في فترة ما قبل ظهور إبداع علي الإطلاق، فلها الميزات التالية:

- **كاملة الهيكل**، يعني أنها لا ينقصها مُكوّنٌ من المُكوّنات الأساسية التي يجب توفرها في أي لغة برمجةٍ تُستخدَم عملياً، مثل:
  - التعامل مع المتغيرات.
  - مُعامِلات حسابية و منطقية logic and arithmetic operators
  - الجُمَل الشرطية.
  - الجُمَل التكرارية.
  - الدوال functions.
  - أوامر إدخالٍ و إخراجٍ للتعامل مع سطر الأوامر console.
  - أوامرٌ للتعامل مع الملفات.
  - المُكوّنات المُركّبة:
    - المصفوفات arrays
    - القواميس dicts

- **تحتوي علي مكتبةٍ تجريبيةٍ بسيطة**، و لها الإمكانيات التالية:
  - التعامل مع الملفات.
  - الإدخال و الإخراج من/علي الشاشة.
  - أوامرٌ رسومية ارسم الخطوط و المستطيلات و الدوائر و النقاط.
  - أوامر التعامل مع الصور (الأطياف حسب تعبير اللغة) كالتحميل و الرسم و الإخفاء و الإظهار.
  - أوامر التعامل مع الأحداث events المدعومة في اللغة، مثل حوادث الماوس و حوادث لوحة المفاتيح و حوادث الأطياف.
  - إمكانيات برمجة الواجهات الرسومية.

- **لها ميزاتٌ متقدمةٌ للغاية عن لغات البرمجة البسيطة**، مثل:
  - دعم البرمجة الكائنية oop بشكلٍ جيد، ففيها:
    - تعريف الفصائل (تكافئ الـclasses).
    - الوراثة inheritance
  - إمكانية تجميع الأكواد المرغوب في إعادة استعمالها في وحدات.

test.k	welcome.k
باستخدام "welcome.k" رحب("سمعان")	وحدة الترحيب إجراء رحب(شخص): اطبع "مرحباً يا "، شخص نهاية

- قويةٌ في جزئية الأحداث و معالجتها events and events handling
- تدعم تنفيذ أكثر من عمليةٍ في نفس الوقت (شيئٌ يشبه في تأثيره تعدد خيوط التنفيذ multithreading).

و للمزيد من التفا صيل و الأمثلة يمكنكم زيارة موقع اللغة.

عيوب لغة كلمات:

و لكن علي الرغم من كل تلك الميزات التي لها، إلا أن كلمات لها عيوبٌ تجعلها لا تُرضِيني علي الإطلاق، و منها:

- هي من اللغات ذات التنويع المتغير dynamically typed، و هذا النوع لايصلح لكتابة البرامج التي تحتاج إلي سرعاتٍ فائقةٍ في التنفيذ، و كذلك يُساعِد في الوقوع في أخطاء التكويد المنطقية لأن اللغة لا تُقيِّد المتغير بنوع بياناتٍ مُعيَّن (يُرجي مُراجَعة هذه النقطة في نقد لغة السوبرنوفا).

- اللغة تحتوي علي مُكوّنين في غاية التشابه هما:
  - الإجراء.
  - الدالة.

  و كان يجب ألا تَضم اللغة مُكوّنين مُتشابهين كهذين علي الإطلاق؛ حيث أن هذا لا داعي له منطقياً، بل و يُسبّب تشابهُّما الشديد إلي البلبلة عند المبتدئين و سنري مثالاً لهذا عند التحدث عن العيب رقم 5، و كان من الأفضل ضم هذين المُكوّنين في مُكوّنٍ واحدٍ أشمل و أقوي من كليهما (و كان يُمكن لكلمات أن تحتوي علي الإجراءات فقط، و لكن مع الاهتمام بأن تأخذ الإجراءات كل جوانب القوة التي تُغْني عن كل مثيلاتها مثلما فعلتُ مع إبداع).

- اللغة تحتوي علي مُكوّني:
  - المصفوفات arrays

- ○ القواميس dicts
  و كان بالإمكان ضم هذين المكونين في مُكوّنٍ واحد، و هذا سيُسهّل علي المبتدئين كثيراً لأنه يضم المكونات المتشابهة التكوين و الوظيفة في مُكوّنٍ واحدٍ أشمل (هذه نقطة نقدٍ فرعيةٍ لكنها موجودةٌ بالفعل).

- الدوال و الإجراءات في كلمات ضعيفة البِنية؛ فهي:
  - ○ تُعيد خرجاً واحداً فقط (في حالة الدالة، أما الإجراءات فلا يفترض أن تُعِيد مخرجاتٍ من الأصل).
  - ○ لا يُمكِن إعطاء قيم ابتدائيةٍ لمُدخَلات الدالة أو الإجراء، و هو الأمر الذي كان بإمكانه تيسير العمل بها بشدةٍ و خاصةً في المشاريع الكبيرة.

- الفصائل في كلمات ثرثارة؛ فلكي تُعَرّف محتويات فصيلةٍ ما فيجب عليك كتابة كلام كثير، و هو علي الرغم من كونه أقل مما هو عليه في حالة لغاتٍ مثل الـEiffel إلا أنه أكبر من المُعتاد في اللغات الأخري (مثل لغة الـpython، و هي اللغة التي تأثّرتْ بها كلمات بشدة).

- مثال1:

```
فصيلة شخص:
    له اسم، سن
    يستجيب ل: اعرض()
نهاية
استجابة شخص ع ل: اعرض():
    اطبع "الاسم هو "، اسم ع، "و السن هو "، سن ع
نهاية
```

مثال2:

```
فصيلة شخص:
    له اسم، سن
    يرد على اكبر.من(أ)
نهاية
رد شخص ش على اكبر.من(الآخر):
    ارجع ب: سن ش > سن الآخر
نهاية
```

- تحتوي كلمات علي أمري "علامة" و "اذهب إلى"، و اللذين يُكافِئان "Goto" في اللغات الإنقليزية، و أنا أرفض هذين الأمرين لأنه من الممكن الاستغناء عنهما و استخدام مُكوّناتٍ أُخري لتؤدي عملهما (الحلقات التكرارية و الجمل الشرطية)، و كذلك فهما يُسبِبان الإرتباك في الكود عند استعمالهما بغزارة (يُرجَي مُراجعة لخطاب Goto statement considered harmful للبروفيسور ديجكسترا، و الذي أرسله للنشر في مجلة (Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148).

```
علامة أ
اطبع "عاوز المصروف يا بابي"
اذهب إلى أ
```

- تعبير الحلقات التكرارية ضعيفٌ للغاية، حيث لا يُتِيح تغيير مقدار الخطوة في العملية إلا بقيمة زيادةٍ مقدارها 1 (و قد نبّه م. محمد سامي إلي أنه ينوي تدارك هذه الجزئية في الإصدارات القادمة بإذن الله عز و جل).

```
لكل أ من ١ إلى ١٠:
  اطبع أ
تابع
```

- المكتبة حتي الآن مبنيةٌ داخل المُفسِر نفسه، و ليست قائمةً بذاتها و ليست مكتوبةً بلغة كلمات نفسها بل بلغة الـ C++ مع مكتبة الـQT.

- لا يُوجَد مُترجِمٌ قويٌ للغة حتي الآن، و كل ما يُوجَد هو مُفسِرٌ يُتِيح لنا استخدام بعض مُكوّنات مكتبة الـQT، و هذا أمرٌ مرفوضٌ لأننا نحتاج إلي كتابة المكتبة الخاصة باللغة كاملةً بها نفسها ليتسني لنا نقل هذا الجزء من العلم البرمجي إلي العربية. و لكن المشكلة أنه لا يُمكِن كتابة مكتبة كلمات بها نفسها كما سيلي في العيب العاشر.

- لا يُمكِن كتابة أوامرٍ بلغة التجميع assembly في برنامج كلمات، و هذا يجعلها غير قادرةٍ علي:
  - إنتاج مكتبتها باستخدامها هي نفسها.
  - إنتاج برامج تعمل علي المتحكمات الميكروئية microcontrollers.
  - كتابة البرمجيات الأخري التي نحتاج فيها إلي التعامل مع الذاكرة مباشرةً، مثل:

- أنظمة التشغيل.
- المُترجِمات compilers.
- المُفسِّرات interpreters.

- الوحدات في كلمات تتيح لنا إنشاء إجراءاتٍ و دوالٍ حرة (أي لا يلزم أن تكون مكتوبةً في فصيلةٍ ما)، و هو ما يجعلنا نعاني من مساوئ نموذج البرمجة الإجرائية في المشاريع الكبيرة، و هذه الصفة تناقض مبدأ الأمن الذي نُريده في لغة البرمجة القوية.

- اللغة لم تستقر في تصميمها بعد، و لم تُوضَع لها خطةٌ تُوضِّح حجمها النهائي المُتوقَّع حتي الآن، ونحن نري بالفعل أن اللغة يزيد حجمها يوما بعد يوم.

- ليس فيها تعبيرٌ لمُعالِجة الأغلاط exception handling.

- الوُصول إلي مُكوِّنات الفصيلة غير مُوَحّد الشكل، فنحن نصل إلي المتغير الداخلي للكائن عن طريق كتابة ا سمه ثم كتابة ا سم الكائن بعده، مثال:

ا سم م = "تا مر"

سن م = ١٢

عنوان م = "شا رع الورد"

حيث م هو كائنٌ object فيه ثلاث متغيراتٍ هي: ا سم، سن، عنوان.

أما الإجراءات و الدوال فيجب أن يُكتَب ا سم الكائن أولًا، ثم نتبعه بعلامة : ثم ا سم الإجراء أو الدالة. مثال:

ق : اضف(١٢)

حيث ق هو الكائن، و اضف هو الإجراء أو الدالة.

- لا تحتوي علي مُكوّنٍ هام للغاية هو الـ enumerations.
- لا تحتوي علي فكرة الـ delegating (الموجو د في لغاتٍ مثل الـ C#).

الحاجة إلي لغة برمجةٍ عربيةٍ جديدة

من خلال كل ما سبق ذِكره من انتقاداتٍ للغات البرمجة العربية الموجودة حالياً: تبيَن لي أنه لا يُمكِنني الإعتماد علي أي واحدةٍ منهن في المشروع الجديد، و ذلك كما قلتُ سابقاً نتيجةً لضعف تلك اللغات أو لإغلاق المُطوِر لمصدرها أو لغيرهما من الأسباب الأخري.

و بصراحةٍ فقد وافق هذا هويً في نفسي؛ لأنه يعني أنني سوف أقوم بتصميم و بناء لغة البرمجة التي أحتاجها من الصفر، و هذا معناه أن الهدف الشخصي في المشروع سوف يحصل علي تقدمٍ هائلٍ و الحمد لله عز و جل.

كما أن هذا سيُريحني بما لا يُقاس فيما بعد؛ فصحيحٌ أنه يعني أنني سأبذل جهداً أكبر بكثيرٍ جداً من الذي كنتُ سأبذله لو كنتُ قد طَوَرْتُ لغةً قديمةً: لكن في النهاية سوف يصير عندي لغة برمجةٍ جديدةٍ صُنِعت بيدي و علي عيني من البداية حتي النهاية، و بالتالي فستَحمِل وجهة نظري في مجال تصميم لغات البرمجة بالكامل بدون أي نقصٍ أو زيادة. و هذا من أكبر المكاسب علي المدي البعيد.

مِنْهاج العمل

وَضعتُ الخطة الزمنية للمشروع الطَّموح منذ الأيام الأولى له، و قد كنتُ كل فترةٍ من الزمن أقوم بالتعديل فيها بما يَتناسَبُ مع ما تعلمتُه من جديدٍ و ما أصبحتُ أراه هو الأهم أو الأقل أهمية. فأضفتُ إليها أشياءاً و أزلت منها أشياءاً أخرى حتى استقرت على الشكل الحالى لها. و على الرغم من اقتناعى الشديد بالجدول الحالى للأعمال إلا أننى أُدرِكُ و أعلم تمام العلم بأن هذا الجدول قد لا يكون نهائياً، بل هو قابِلٌ للتعديل فيه بالإضافة أو النقصان.

و الجدير بالذِّكْر أن الجدول كان له شكلٌ مُختلِفٌ تماماً حينما بدأتُ العمل علي المشروع فعلياً، و لكن الأمور كلها اختَلَفت بعدما تأهلتُ في مُسابَقة برنامج (نجوم العلوم) في موسمه الرابع؛ حيث اضطررتُ ساعتها أن أُخالِف ما كنتُ قد اقتنعتُ به بعد فترةٍ من الدراسة و التفكير، و غيَّرتُ أسلوب العمل بشكلٍ جذري ليتناسب مع ظروف مُشاركتي في المُسابَقة !

و علي الرغم من أني لم أُكمِل حتي المراحل الغاية في التقدم في المُسابَقة إلا أنه بحمد الله تعالي كان في ذلك التغيير الخير كل الخير، و لو كنتُ أعلم أن الأمر سيكون بهذا الشكل لكنتُ سلكتُ تلك الطريق منذ البداية، و لكن لا يعلم الغيب إلا الله عز و جل، فله الحمد في الأولي و الآخرة.

و قد أدرجتُ الخطتين في هذه الرسالة: خطة ما قبل نجوم العلوم، و خطة ما بعد نجوم العلوم؛ حتي يَتَسَنَّي لأهل الإختصاص (و كذا كل من يَرغَبُ) الإضطلاع عليهما لمُلاحظة الفارق و استخلاص الفوائد.

و الخطة الأخيرة تُوضِّح الخطوات الزمنية الكاملة للمشروع منذ بدايته و حتى المراحل المُتقدِّمة منه، و التى يكون بعدها قد وَصَل إلى مرحلة البلوغ التى نرجوها له، و حينها يكون قادراً بمُجتَمعه الضخم و مُنتَجاته الثرية على طَرْق أبواب الجديد فى العلوم و المجالات البرمجية مما لم نُخطِّط له من قبل بمُنتَهى القوة و الحرية بإذن الله عز و جل.

و بالطبع فإن هناك خطواتٍ قد تم الإنتهاء منها و نُفِّذت فعلياً على أرض الواقع، و لكن حتي هذه لم أشطبها من قائمة الأعمال؛ لأننى أريد من الجميع أن يعلموا كل الخطوات التى تخص المشروع سواءٌ أكانت قد نُفِّذت أم لم تُنفَّذ بعد حتى يكونوا على بينةٍ من الأمر، و يكون بمقدرتهم تكوين نظرةٍ شاملةٍ عن المشروع و المرحلة التى وَصَل إليها.

و لكنى وَضعتُ علاماتٍ تدل القارئ عن كَوْن كل مرحلةٍ قد نُفِّذت أم لا، فالمراحل التى تم الإنتهاء منها وَضعتُ أمامها علامة (✓)، و التى خَلَت من وُجود العلامة أمامها هى المراحل التى لم تُنفَّذ بعد و يُنتَظر تنفيذها فيما بعد بتوفيق الله تعالى.

خطة (ما قبل نجوم العلوم)

- وضع قواعد لغة إبداع و التصميم المبدئى لها.
- تصميمٌ شاملٌ لأقسام المكتبة القياسية و تحديد جزء المكتبة الخاص بالإصدارات الأوّلية منها.
- صنع مُترجِمٍ كاملٍ لإبداع بلغة الـjava:
  - تصميم و بناء الواجهة الأمامية front end و التى تتكون من الماسِح scanner و السّابِر parser.
  - تصميم و بناء المُحَقِّق المنطقي semantic analyzer.
  - تصميم و بناء مُنتِج التمثيل الوسيط IR.
  - تصميم لغات التجميع العربية للمُعالِجات التى سيتم دعمها كمرحلة أولى فى المُترجِم.
  - تصميم و بناء مُولِّدات الأكواد التى ستُحوّل كود الشفرة الوسيطة إلى لغة التجميع الخاصة بالمُعالِجات المَعنيّة.
  - تصميم و بناء المُجَمِّعات assemblers التي ستُحوِّل أكواد التجميع إلى أكواد الآلة المعنية.
  - تصميم و بناء مُحَسِّن شجرة السبر parsing tree optimizer.
  - تصميم و بناء مراحل التحسين فى الكود الوسيط IR optimizing.

  و إنتاج البرامج سيكون لبيئة القنو/لينوكس gnu/linux فقط فى البداية.
- بناء أساسيات المكتبة لمُعالِجات X86.
- بناء الأصناف الأعلى فى المكتبة اعتماداً على المُكوِّنات البسيطة الأولى الموجودة بالفعل. و المُكوِّنات التى ستُكتَب فى البداية هى الخاصة بالإصدارة رقم (1.0) من المكتبة القياسية.
- تطوير المُترجِم بحيث يكون قادراً على إنتاج برامج تعمل على نظام التشغيل windows.
- تصميم آلة إبداع الوهمية التي يُختَصر اسمها لـ آبو.
- برمجة بِيئة الـ آبو.
- عَرْض الأمر على الآخرين و طلب مساعدتهم.
- دعم المزيد من أرضيات العمل للمُترجِم و المكتبة القياسية من المُعالِجات الأكثر شهرة و انتشاراً. على سبيل المثال المُعالِجات التالية:
  - ARM
  - AMD
  - Sun sparc
  - MIPS

- تطوير المُترجِم بحيث يكون قا دراً على إنتاج برامج تعمل على نظام التشغيل ماك mac.
- تطوير المُترجم بحيث يكون قا دراً على إنتاج برامج تعمل على نظام التشغيل سُولارِز solaris.
- التطوير فى الشفرة المصدرية لمُترجِم إبداع لإنتاج إ صداراتٍ جديدةٍ منه با ستخدام لغة إبداع، و كلما أنتجنا إ صدارةً جديدةً تركنا القديمة و ا ستخدمنا الجديدة لإنتاج القا دمة، و كذا لإنتاج المُفسِّر و بيئة آبو و هكذا،

و فائدة هذا:

- أن النظام سيكون مُحتوياً لنفسه؛ فالبيئة البرمجية مكتوبةٌ بذات اللغة التى خُصِّصت لها و هذا من أكبر الإثباتات على قوة اللغة و احترافيتها.
- سيجعل مُطوِّري إبداع مُستخدِمين لها و بالتالى يُعطِيهم فر صة المُمارسة العملية لها على أو سع نطاق، و كذا مُلاحظة نقاط ضعفها و جوانب النقص فيها و التى تحتاج إلى مُعالَجة.
- سيُعطِى هذا كنزاً ضخماً للمُتعلِّم الجديد للُّغة حينما يَجشد أمامه شفرةً ضخمةً يستطيع من خلال قراءتها أن يتعلم كيفية البرمجة الإحترافية با ستخدام اللغة الجديدة.
- سيُحوِّل هذا الأمر العلمَ البرمجى الخاص بإنتاج مُترجمات لغات البرمجة إلى علم يعتمد على اللغة العربية فى أدق دقائقه و هى الكود البرمجى نفسه، و بالتالى فإن التعريب حينها سيكون أمراً مفروغاً منه و لم يتبق منه إلا الشُروحات فقط.
- سيُعطِى هذا المُتعلِّمين الجدد للُّغة و الراغبين في التعلم العملي علي المشاريع الضخمة فر صةَ المشاركة فى تطوير بيئة البرمجة و هكذا نكسب جهود الكثيرين.

و بالتوازى مع تطوير البيئة البرمجية سنُطوِّر فى المكتبات الخا صة باللغة أيضاً و لكن على نحوٍ أقل قوة.

- سينقسم فريق العمل إلى قِسمَين:
  - القِسْم الأول يُتابِع تطوير البيئة البرمجية.
  - القِسْم الثانى يُتابِع التطوير فى المكتبات المُختلِفة للُّغة.
- تصميم لغة قواعد البيانات العربية.
- إنتاج لغة قواعد البيانات العربية و نظام إدارتها.

خطة (ما بعد نجوم العلوم)

- وضع قواعد لغة إبداع و التصميم المبدئى لها.
- تصميمٌ تجريبيٌ شاملٌ لأقسام المكتبة القيا سية و تحديد جزء المكتبة الخاص بالإ صدارات الأوّلية منها.
- صنع مُفسِرٍ كامل لمعظم المُوا صفات القيا سية لإبداع بلغة الـjava) بحيث يتم تصحيح البرامج ثم تنفيذها مبا شرة بدون إنتاج ملفات أكوا د و سيطة مثل الـjava bytecode) و أ سميتُه **أبْدِع** :obde3

- تصميم و بناء الواجهة الأمامية front end و التى تتكون من الفاحص scanner و السابرparser.
- تصميم و بناء المُحقِّق المنطقي semantic analyzer.
- تصميم و بناء عُقَد التنفيذ executing node.

- تكملة و نشر مرجع المشروع (رسالة البرمجة بإبداع).
- بناء الإ صدار الأول من المكتبة.
- عَرْض الأمر على الآخرين و طلب مساعدتهم.
- تكملة بناء بقية المُوا صفات القيا سية لإبداع، مع:
- التو سع في بناء الأ صناف الأعلى فى المكتبة اعتما داً على المُكوّنات البسيطة الأولى الموجودة بالفعل.
- تحويل (المُفسِر) إلي (مُفسِرٍ و مُترجِم) بحيث يمكنه إنتاج برامجٍ أ صليةٍ native تعمل علي المُعالِجات مبا شرة، و سيكون إنتاج البرامج في البداية لنظام تشغيل القنو/لينوكس GNU/linux مع عائلة مُعالِجات X86.
- تطوير المُترجِم بحيث يكون قا دراً على إنتاج برامج تعمل على نظام تشغيل الويندوز.
- تطوير المكتبة بحيث تتوافق مع هيكلية المُفسِر/المُترجِم.
- تصميم اللغة الو سيطة IR لآلة إبداع الوهمية آبو.
- برمجة بيئة الـ آبو.
- إعا دة برمجة المُفسِر بحيث يحتفظ بخا صية تنفيذ الأكوا د انطلاقاً من الملفات النصية مبا شرةً، بالإضافة لخا صية إنتاج كو د البرنامج بلغة آلة إبداع الوهمية آبو في البداية، و من ثم يُمكِن تنفيذ هذا الكو د بعدها علي الآلة الوهمية علي أي نظام تشغيل مدعوم.
- دعم المزيد من أرضيات العمل للمُترجِم و بناء أ سا سيات المكتبة للمزيد من المُعالِجات المشهورة. علي سبيل المثال المُعالِجات التالية:

- ARM
- AMD

- Sun sparc
- MIPS

- تطوير المُترجِم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل الماك.
- تطوير المُترجِم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل السُولارِز.
- تطوير المُترجِم بحيث يكون قادراً على إنتاج برامج تعمل على نظام تشغيل اليونيكس.
- التطوير فى الشفرة المصدرية لمُترجِم إبداع لإنتاج إصداراتٍ جديدةٍ منه باستخدام لغة إبداع، و كلما أنتجنا إصدارةً جديدةً تركنا القديمة و استخدمنا الجديدة لإنتاج القادمة، و كذا لإنتاج المُفسِّر و البيئة آبو، و قد سبق التَّوسع في شرح هذه الخطوة و فوائدها.

- سينقسم فريق العمل إلى قِسْمَين:
  - القِسْم الأول يُتابِع تطوير البيئة البرمجية.
  - القِسْم الثانى يُتابِع التطوير فى المكتبات المُختلِفة للُّغة.
- تصميم لغة قواعد البيانات العربية.
- إنتاج لغة قواعد البيانات العربية و نظام إدارتها.

الخطوط العريضة لإدارة المشروع

لم يتم التحدث في النقاط السابقة عن أمورٍ في غاية الأهمية، هن:

- كيفية الحصول علي التمويل الكافي للإستمرار في العمل،
- المرحلة التي سيتم فيها جعل المشروع مفتوح المصدر،
- الهيكل الإداري للمشروع و كيفية التعاون بيني و بين من يرغب في المشاركة فيه،
- كيف يُمكِن لمن يرغب في المساعدة أن يفعل هذا.

لذا ففي هذا الجزء سيتم شرح هذه المسائل باختصارٍ بإذن الله عز و جل.

أطوار المشروع:

سيكون للمشروع طَوْران من أطوار الحياة، الطَوْر الأول هو: طَوْر العمل الفردي، و الطَوْر الثاني هو: طَوْر العمل المُؤسَّسي الجماعي،

- **الطَوْر الفردي**

المرحلة الأولي من حياة المشروع، و فيها ستكون مُنتجاته مُغلقة المصدر و سيكون العمل علي تطويرها فرديا تماماً؛ حيث سأكونُ المُبرمِجَ الوحيد فيه، و الوحيدَ الذي يَتعامل مع التساؤلات و/أو الإقتراحات و/أو الدعايا و/أو النقاشات المُختلفة لمن يهتمون بالمشروع (سلباً أو إيجاباً).
و سيتم تمويل المشروع عن طريق التبرعات[6] من الأفراد و المُؤسَّسات، و شراء المنشورات الورقية الخاصة بالمشروع (مثل هذا الكتاب).

- **الطَوْر المُؤسَّسي**

و فيه سيتم تحويل المشروع من عملٍ فردي إلي مُؤسَّسةٍ لها كيانٌ خاصٌ بها، و ذلك لرعاية المشاريع مفتوحة المصدر التي ذُكِرت من قبل في غايات مشروع البرمجة بإبداع، و سيتم قبول مُشارَكات من يَرغب في تطوير الاكواد المصدرية لمُنتجات المؤسسة، مع الأخذ في الاعتبار أن أسماء المُنتَجات و شعاراتها سيتم اعتبارها علاماتٍ تجارية (تماماً كما يحدث في حالة نواة اللينُوكس linux kernel).

أما التمويل فسيَكون مُعتمِداً علي الطرق التالية:

- التبرعات

6 لمعرفة كيفية التبرع للمشروع يُمكِنكم زيارة الموقع الرسمي له: http://www.ebda3lang.blogspot.com/p/blog-page_2244.html

- شراء المنشورات الخاصة بالمشروع، مثل كتب تعليم البرمجة بلغة إبداع، و كتب تعلم التعامل مع المُفسِّر و المُترجم و بيئة البرمجة المتكاملة و غيرهن من المُنتَجات الأخري.
- بيع خدمات التنزيل installing و الصيانة maintaining للمُؤسَّسات التي تَرغب في استخدام المُنتَجات البرمجية للمشروع. سواءٌ أكانت تلك المؤسسات تعليمية أم غير تعليمية.
- بيع خدمات التدريب للأفراد، و للمُؤسَّسات التي تَرغب في تدريب مُوظَّفيها علي المُنتَجات البرمجية للمشروع، و كذا إجراء امتحاناتٍ تقويميةٍ لإعطاء شهاداتٍ مُعتمَدةٍ من المُؤسَّسة لمن يَرغب في ذلك.

موعد الإنتقال من الطَوْر الفردي إلي الطَوْر المُؤسَّسي:

هذا الأمر يعتمد بشكلٍ كاملٍ علي ما يتوفر من تمويل، فإذا كان هناك عجزٌ ماديٌ شديد: فسيأخذ الأمر فترةً غير صغيرةٍ للتطور، أما لو توافرت الموارد المادية فسيكون بالإمكان الإسراع بهذه الخطوة بشكلٍ كبيرٍ يتناسب مع مقدار الدعم الموجود.

كيفية أخذ القرار في المُؤسَّسة فيما بعد:

هذا أمرٌ سابقٌ لأوانه، و لكن ما يُمكِن تأكيده أن أخذ القرارات في كل مشروعٍ من المشاريع المُختلِفة سيكون بمنهج التشاور الواضح بين أفراد مُجتمَع المُطوِّرين له، مثلما يَحدث في مُجتمَع نواة اللينوكس (بل ربما نعتمد علي ذات الآليات التي يعتمدون عليها في عملهم). و لو حَدَث أن انفرد مالك العلامة التجارية للمشروع بقراراتٍ لا تراها الأغلبية: فستظل لهم القدرة علي الإنفصال و إنشاء مشروعٍ مُوازٍ مبني علي آخر الإصدارات التي وصل إليها المشروع، و هذا بفضل الرخصة المفتوحة التي ستعتمد عليها المُؤسَّسة في عملها بإذن الله عز و جل.

هل ستقتصر المُؤسَّسة علي البرمجيات فقط في المستقبل ؟

هناك نيةٌ صادقةٌ لَدَيَّ لتحويل المؤسسة إلي ما يُشبِه شركة apple في عالم الحوسبة، بحيث تكون قادرةً علي إنتاج أنظمةٍ مُتكامِلةٍ بكل ما تشمله من برمجياتٍ و عَتادٍ صلبٍ hardware تعمل عليه تلك البرمجيات.

و لا أقصد بالعتاد الصلب هنا أجهزةَ الحاسوب العادية فقط، بل أعني كذلك كل الأجهزة الرقمية القابلة لإعادة البرمجة، سواءٌ أكانت حواسيباً أو هواتفاً جوَّالةً أو حتي أجهزة تحكمٍ رقمي قابلةٍ للبرمجة plc، أو غيرهن.

علي أن هذا قد يستغرق فترةً طويلةً حتي يصير حقيقةً ملموسة، لما يحتاجه من متطلباتٍ تكاد تجعل الأمر يدخل في عداد المستحيل، لكن علي أية حالٍ: النية موجودةٌ و جاهزةٌ للتحويل إلي واقعٍ متي ما أمكن فعل ذلك.

القاموس

بعض الإصطلاحات العربية المُستَخدمة ليست هي الأشهر في ترجمة مُرادِفاتها التقنية الإنجليزية، و إنما استخدمتُها لأنها أدق و أوضح من حيث المعني، بينما الترجمة المشهورة حَرْفِيةٌ أكثر من اللازم.

الإصطلاح العربى	الإصطلاح الإنقليزى المكافئ
برمجة على مستوى الخانة	Bitwise programming level
برمجة كائنية	Object oriented programming
بيئة برمجة متكاملة	Integrated development environment (IDE)
جيب الزاوية (جا)	Sin
جيب تمام الزاوية (جتا)	Cos
خوارزم	algorithm
خيوط التنفيذ	Executing threads
طابور	queue
ظل الزاوية (ظا)	tan
قائمة	list
قائمة مرتبة	Sorted list
كائن = نسخة	object = instance
كود أصلى	native code
كود محكوم	managed code
مُتحكِّمات ميكروئية	Microcontrollers
مُترجِم	compiler
مُعالَجة الأغلاط	exceptions handling
مُفسِّر	interpreter
مَقرُوئية	readability
مَكتُوبِية	writability
مُكدِّس	stack
نظام تشغيل	Operating system
نُواة (في نظام تشغيل)	kernel
واجِهة	interface

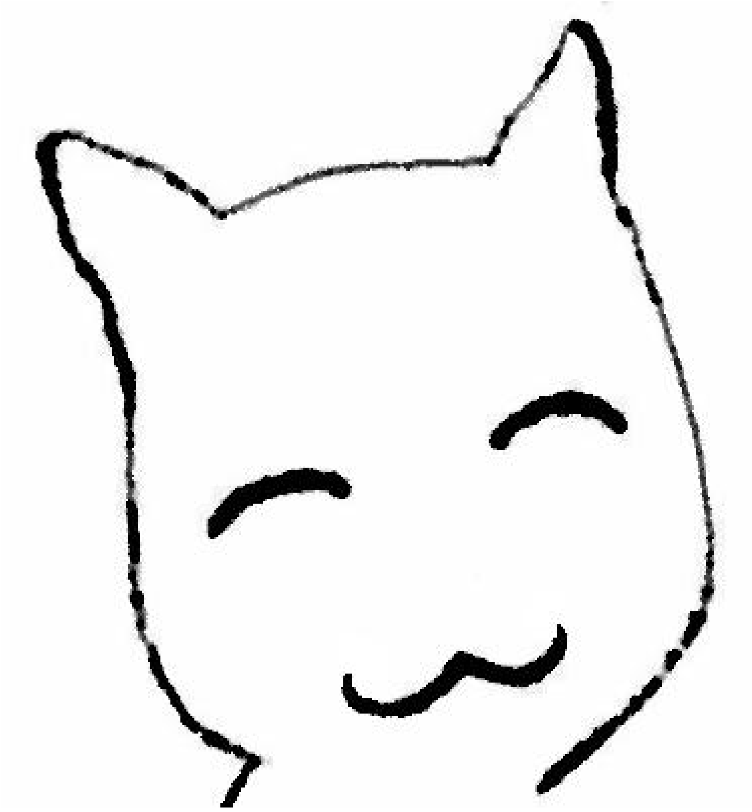
inheritance	وِراثة
debugger	مُنقِّح
Free software	برمجيات حرة
Programming language theory	نظرية لغات البرمجة
Application programmer interface (API)	واجهة مُبرمِج التطبيقات
bug	عِلّة
unix-like	شبيه اليُونِكس
Assembly language	لغة التجميع
processor	مُعالِج
assembler	مُجمّع
Programming paradigm	منهج برمجي
scanner	ماسِح
parser	سابِر
readability	مقروئية
writability	مكتوبية
Semantic analyzer	مُحقِّق منطقي
Intermediate representation (IR)	تمثيل وسيط
Object code	الكود الهدف
optimization	تحسين
productive	مُنتِج
List box	صندوق قائمة
Arithmetic operator	مُعامِل حسابي
Logic operator	مُعامِل منطقي
function	دالة
console	سطر أوامر
array	مصفوفة
dict	قاموس
event	حَدَث
class	فصيلة = صنف
Events handling	مُعالَجة الأحداث

multithreading	تعدد الخيوط
Dynamic typing	تنويعٌ مُتغيِّر
Front end	واجِهة أمامية
computational thinking	التفكير الحَوْسَبِي

المصادر

- سُنَن الترمذي رحمة الله عليه.
- Wikipedia the free encyclopedia.
- مقدمة كتاب (العِبَر، و ديوان المُبتدأ والخَبَر، في أيام: العرب، و العَجَم، و البَرْبَرْ، و مَن عا صَرَهم مِن ذوي السلطان الأكبر) لابن خلدون رحمة الله عليه، و تُعرَف اختصاراً بـ(مقدمة ابن خلدون).
- مقال (مقارنة بين ثلاث لغات برمجية عربية جيم، زاى، لوجو) لكاتبه : طه زروقي.
- العرض التقديمي المُسمَّي (تطوير بيئة لغة البرمجة العربية APL) من إعداد: أيمن بن فهد السند و سعد بن عبدالله السلامة.
- (Arabic programming languages) by: sami sarhan.
- العرض التقديمي المُسمَّي (تعريب لغات البرمجة (لغة ج)) من إعداد: فيصل الزامل و عبدالله الغنيم، تحت إشراف الدكتور: عبدالملك السلمان.
- الموقع الرسمي للغة (ج): http://jeemlang.com/
- العرض التقديمي (لغة البرمجة العربية خبير)، جامعةالملك فهد للبترول و المعادن، كلية هندسة و علوم الحاسب، من إعداد: فهد علي القحطاني.
- كتاب (لغة خوارزمي للحاسب الإلكتروني)، تأليف: عبد الفتاح جمال عبد الحفي، الطبعة الثانية 1406هـ -1986م، شركة الرائد للحاسبات الإلكترونية.
- الموقع الرسمي للغة (زاي): zegour.uuuq.com
- كُتيِّب (كتابة مبرمج لغة ض)، تأليف: خليل الأمين عبد الجواد.
- كتاب (Design of Arabic programming language)، تأليف: عبد العظيم أحمد عمّوري، إشراف: لبنى بدري فهد، Philadelphia University Faculty of Engineering Department of Computer Engineering
- كُتيِّب (CATIB:A Bilingual Authoring Language with Hypertext and Multimedia Capabilities)، تأليف: صبري عبد الله محمود و محمد محمود مندورة، في (مجلة الحاسوب السعودية) المجلد 6 العدد 6
- كُتاب (بيئة البرمجة "كاتب" و تطبيقاتها في تطوير نظم تعليم حاسوبية ذكية)، تأليف: صبري عبد الله محمود و محمد محمود مندورة. في (الندوة الثانية لتعريب الحاسوب، جامعة الملك سعود، شَوَّال 1414 هـ).
- كتاب (لغة البرمجة سوبرنوفا) الإلكترونى العربى للمهندس: محمود سمير فايد.

- شرائح العرض الإنقليزية المُعنوَنة (Supernova_Abstract) و المُضَـمَّنة مـع ملفـات الإ صدار رقم 1.2 من لغة البرمجة سوبرنوفا للمهندس: محمو د سمير فايد.
- كتـاب (Supernova Programming Language) الإلكـترونى الإنقليـزى للمهندس: محمو د سمير فايد.
- العرض التقديمي (كلمات: لغة البرمجة العربية الجميلة)، لـ م. محمد سامي.
- كُتيّب (تعلم كلمات بالأمثلة) تأليف: م. محمد سامي.
- الموقع الرسمي للغة (كلمات): www.kalimat-lang.com
- كتاب (البرمجة باللغة العربية)، تأليف: أ. عمر مكدا شي. شركة منشورات: دار الراتب الجامعية،
- موقع الشبكة: http://itwadi.com و المُشارَكات و المُناقَشات التى دارت عليه حـول إنتاج لغة برمجةٍ عربية.
- كتاب (Visual Basic للجميع) الإلكترونى للمؤلف: تركى العسيرى.
- عدة مواقعٍ شبكيةٍ أخرى للأسف لا أذكرها بالضبط و إن كنتُ قد استفدتُ منها علمياً.



القسم الثاني:

عن المواصفات القياسية للغة

البرمجة العربية إبداع

الإصدارة (1.0) من اللغة

شرح المُواصَفات القياسية لإبداع

هذا الجزء ليس على هيئة كتب المَعايير القِيا سِية المشهورة رغم أنه وُضِع للقيام بما يُقارِب هذا الدور، و ليس على شكل كتب الشروح التعليمية للغات البرمجة التى تُدَرَّس أو تُباع للمُبرمِجين العا ديين الراغِبين فى تعلم اللغة الجديدة رغم أنه أيضاً وُضِع للقيام بما يُقارِب هذا الدور؛ فهو يَمزِج بين الشكلين السابقين و يَصلُح للقِيام بدوريهما بكل كفاءة.

فمن ناحية المعايير القيا سية سوف تكون كل القواعد مشروحةً بلُغةٍ واضحةٍ كل الوُضوح لتُحدد ما المسموح به فى اللغة و ما غير المسموح به، و ما هى شروط صحة كل عملية (إذا كان لها شروط صحةٍ معينة).

و لكن رغم ذلك فإن أسلوب الشرح غير جافٍ كما هو فى كتب المعايير القِيا سِية للغات البرمجة، بل هو سَلِسٌ سهل التناول و مدعومٌ بكم كبيرٍ من الأمثلة التى تُوضِّح القواعد عملياً، و بالتالى فهذا الجزء يَصْلح ككتابٍ تعليميٍ للمُحترِفين فى لغاتٍ أخرى و الراغبين فى تعلُّم لغة إبداع كاملةً بسرعةٍ كبيرةٍ و بمنتهى الكفاءة كذلك،

و أقولُ **المُحترِفين** لأنه يُشترَط في الراغب فى تعلم إبداع من خلال هذا الجزء أن يكون على درايةٍ بكل المواضيع التى نتحدث عنها فيه؛ حيث أنه لا يشرح أصل الموضوع بتمعن، بل يتطرَّق إلى موقف لغة إبداع منه فقط و يستفيض فى شرح هذا الموقف. و في الفترة القادمة سيكون هناك كتابٌ آخرٌ بإذن اللهـ تعالي خاصٌ بتعليم البرمجة للمبتدئين من الصفر حتي الاحتراف و فيه سأقوم بشرح كل قواعد إبداع باستفاضةٍ و توسع.

و كذا سيكون هناك فيما بعد كُتَيِّبٌ منفصلٌ بإذن الله عز و جل أُقدِّم فيه طريقةً جديدةً لو صف نَحْو لغات البرمجة و قواعدها غير الطرق المعتادة المعروفة حالياً، و هى فى نظرى أفضل من تلك الطرق المعتادة بكثير (على الأقل بالنسبة لى)، و في ذات الكُتَيِّب سأقوم بشَرح تلك الطريقة الجديدة بمُنتهَى الإستفاضة و الشمولية ليرجع إليه القارئ الكريم عند الرغبة فى فهم أساسيات الطريقة الجديدة لو صف نَحْو لغات البرمجة.

و هذا الجزء سيتضمن قواعد الإصدارة الأولي فقط من اللغة، فلأن اللغة لم يكتمل بناؤها بعد فمن المُمكِن أن تَحدُث بعض التغييرات في أجزاء التصميم الموجودة حالياً و التي لم تُنفَّذ بعد، و كذا فقد قمتُ هنا بالتعريف بالمُكوِّنات التي لم يتم الإنتهاء من بنائها بالفعل في الإصدار الحالي من المُفسِّر القياسي المُسمَّي: **أبْدِع**.

صفات اللغة:

هى لغة

بسيطة: عدد قواعدها كأقل و أسهل ما يكون، و يكفى كتابٌ مُتوسِّط الحجم لتغطية نَحْو اللغة بشكلٍ كامل،

سهلة و تعليمية: يُمكِن استخدامها كلغةٍ تعليميةٍ فى مجال البرمجة للأطفال الصغار بسهولةٍ و سرعة، فلقد رُوعِى هذا الجانب بشدةٍ أثناء تصميمها،

هجينة: لها صفاتٌ مُشترَكَةٌ تَمزِج بين صفات اللغات الإجرائية و لغات البرمجة الكائنية، و تُحاوِل استخلاص الأفضل من كل نوعٍ من النوعين،

قوية: لأنها تُراعِى القوة فى كل مَناحى تصميمها،

آمنة و نظيفة: تختار للمُبرمِج أفضل الإختيارات و الأدوات و تُتِيحها له ليستخدمها فى برامجه،

متعددة الخيوط: أى بإمكانها إنتاج برامج ذات خيوط تنفيذٍ مُختلِفة،

تفسيرية و مترجمة: أى يُمكِنها أن تُنتِج كوداً محكوماً managed code و كوداً أصلياً native code باستخدام نفس النص البرمجى الإبداعى و نفس المكتبات،

مفتوحة المصدر: فصناعة مُترجِم أو مُفسِّرٍ لها أمرٌ مُتَاحٌ للجميع حسب التوضيحات الواردة فى الرخصة التى تلى هذه الصفحة، فيُرجَى الإنتباه لهذه الوثيقة الفائقة الأهمية،

مستقرة: فلا نية هناك على الإطلاق للإضافة إليها إلا بمُعدَّلاتٍ بسيطةٍ جداً و عند الحاجة الماسَّة لذلك، و على العموم فالتغيُّرات ستكون قليلةً للغاية إن حدثت.

رخصة مُوا صَفات لغة البرمجة (إبداع) فى الإصدارة (1.0):

تخضع مُوا صَفات لغة البرمجة العربية إبداع فى إصدارتها رقم 1.0 للبُنود التالية:

- يُمكِن لأى أحدٍ أن يُطالِع الموا صفات الخا صة باللغة و أن ينشرها دون أى قيودٍ أو عوائق ما دام الأمر خارج نطاق الإستغلال التجاري، و لكن يجب الحصول علي إذن مُصمِّم اللغة إذا كان النشر تجارياً.
- الإسم العربي للغة (إبداع) و كذلك الإسم الإنقليزي (ebda3) و اسم المُفسِّر الخاص بها بالعربية (أُبْدِع) و ترجمته الإنقليزية (obde3) تُعتبَر علاماتٍ تجاريةٍ لمُصمِّم اللغة و لا يجوز التعدي عليها.
- صناعة مُترجِم compiler أو مُفسِّرٍ interpreter لها أمرٌ مُتَاحٌ للجميع بشرط بناء كافة المُوا صَفات الخا صة بها في هذه الإصدارة.

مبادئ التصميم التى تم اعتبارها أثناء تصميم إبداع:

- استخدام الكلمات و الرموز المألوفة، و تقليل الرموز و العلامات التى لا تنتمى إلى الحروف العربية قدر الإمكان لتكون سهلة التعلم و الإستيعاب.

- تحويل النص البرمجى إلى ما يُقارِب الوصف العادى باللغة العربية الحديثة، حتى أن أى شخصٍ غير مُتَخصِّصٍ يُمكِنه فهم الكثير بمُجرّد قراءة البرنامج المكتوب بإبداع. و لكن مع المُحافَظة على البُعد عن الحشو الموجود فى اللغة العادية و الذى لا يُناسِب صِفتَي الإنتاجية و المكتوبية الكبيرتين للُّغة.

- مزج التعبيرات و المُكوّنات البرمجية المُتقارِبة فى مُكوّنٍ أو تعبيرٍ واحدٍ أشمل و أعم لتكون كذلك سهلة التعلم و الإستيعاب. و لتفادى البلبلة و الخلط الناجِمين عن التشابه الصِياغى بين تلك المُكوّنات أو التعبيرات.

- الاستغناء عن أى مُكوّنٍ أو تعبيرٍ يزيد من قواعد اللغة بدون تقديم فائدةٍ يتميز بها و يكون الوُصول إليها بالمُكوّنات الأخرى صعبا للغاية.

- الإستغناء عن أى مُكَوّنٍ أو تعبيرٍ يُدخِل المبرمج المُستخدِم للّغة فى تفاصيلٍ فنيةٍ تخص المُترجِم أو نظام التشغيل أو العَتَاد الصلب، أو بطريقة تنفيذ البرنامج بشكلٍ عام. و مثل ذلك الـ Preprocessor الخاص بعائلة الـC (مثل الـC و الـ#C و الـ++C) و الذى لم نأخذ منه إلا أقل القليل.

و الطريقة الوحيدة التى يُمكِن للمُبرمِج بها الوصول إلى مثل تلك التفاصيل هى الأدوات المُتوفّرة فى المكتبة القياسية لإبداع، و بالطبع فإن هذا يُحافِظ على انتهاج المُبرمِج نهجاً برمجياً أفضل.

- جَمْع الأفضل من كل لغة برمجةٍ و عدم الإهتمام بتصنيفٍ مُعيّنٍ للّغة يحد من صفاتها و ميزاتها، كأن نجعلها لغة كائنية صرفةً أو لغةً وظائفية صرفةً فنُحَمّلها عيوب النوع الذى خُصّصَت له و نُفْقِدُها ميزات النوع الذى لم تنتم له، و ذلك بالطبع عند القُدرة على الجَمْع (و هى المُتوافِرة فى الغالب) أما عند عدم القُدرة على الجمع فيتم اختيار الخاصية الأفضل و الاكثر انسجاماً مع باقى قواعد اللغة.

- عدم الإعتراف بالمبدأ القائل بأن مُصمِّم اللغة ليس عليه أن يبتكر أ شياءً جديدةً بل هو مُجرد جامع للصِّفات الجَيّدة فى لغات البرمجة الموجودة فعلاً فى اللغة الجديدة، و بالتالى اعتماد أى صِيغةٍ أوٍ فِكرةٍ يتم الإقتناع بأحقيتها و جدارتها بالضم للُّغة الجديدة.

- مُساعَدة المُبرمِج على إنتاج كودٍ آمنٍ عن طريق:
  1- مُعالَجة الأغلاط.
  2- تَعبيراتٌ و مُكوِّناتٌ قويةٌ تُساعِده على التغلب على الأجزاء الصعبة من مهمة التكويد و تُوفِّر عليه العناء.
  3- مبدأ التوارث الذي يُعطِى المُبرمِج القدرة على الإستفادة من جهد الآخرين و إنتاجهم الذى تم التأكد من أمنه.

قائمة اللغات التى تم الإستفادة منها عند تصميم إبداع:

فى كل مَرّةٍ يتم فيها الإعلان عن لغة برمجةٍ جديدةٍ تكون هناك تلك المقولة التى مللناها من كثرة تردادها و هى أن: المُصمِّم **"إختارَ أحسنَ ما فى اللغات الأخرى و وَضَعَه فى اللغة الوليدة"**، و فى مُعظم الأحيان حينما نُدقّق فى الأمر نرى أنه يُخالِف أغلب توقعاتنا، حتي نَشُك في أنه لم يكن إلا فقاعةً إعلاميةً فارغة ليس إلا !، و أن الهدف الوحيد من ورائه كان دفع الناس للإلتفاف حول اللغة الجديدة و عقد المُقارَنات بينها و بين اللغات القديمة فيكون انتشارها أسهل و أقوى.
لكني أجد نفسي غير قادرٍ علي التهرب من فِعْل نفس الشيء هنا؛ فأنا بالفعل حاولتُ قدر الإمكان إعطاء إبداع الميزات التي في اللغات الأخري و تلافي عيوب تلك اللغات بحيث تُصبِح بالفعل اللغة الجامِعة للصفات التي ينبغي أن تكون في لغات البرمجة عالية المستوي عامّة الأغراض.

و فيما يلى قائمةٌ بلُغات البرمجة الموجودة بالفعل و التى استفدتُ منها بشكلٍ واضحٍ أثناء تصميم إبداع: إما معرفةً لمُكوّناتٍ مُفيدةٍ تُضَم إليها، و إما معرفةً بمُكوّناتٍ سيئةٍ يجب رفض ضمها إلى أى لغةٍ برمجية، كما سيتضح من الأجزاء التالية لجزء و صف اللغة فى هذه الرسالة و التى تُناقِش القرارات التصميمة المُختلِفة التى اتخذتُها أثناء التصميم.
ففى مكانٍ تالٍ فى هذه الرسالة سوف أتطرق بإذن الله تعالي إلى شرح قَناعاتى المُختلِفة فى مجال تصميم لغات البرمجة بشكلٍ مُستفِيض، و من ثم أتطرق منها إلى ذِكْر الدوافع المنطقية التي جعلتني أتخذ القرارات التصميمية التي جَعَلَتْ إبداع علي الشكل الذي هي عليه الآن، و في مكانٍ آخر سوف أذكر المُكوّنات التى تُوجَد فى لغات برمجةٍ أخرى و تشتهر بها و رفضتُ ضمّها إلى إبداع و أسباب ذلك الرفض، و ذلك كتوضيحٍ تامٍ لتأثرى بتلك اللغات و إن كان ذلك التأثر قد حدث بصِيغةٍ سالبةٍ لا موجبة.

قائمة اللغات:

- C#
- Java
- Matlab
- C++
- Visual basic
- F#
- ج

- Fortran
- Ada
- Pascal
- python
- Eiffel
- C

هَيْكَل البرنامج فى اللغة

سوف أَذكرُ فى هذا الفصل التكوينَ العام للبرنامج المكتوب بإبداع بكل ما يُمكِن أن يحتويه من حيث المُكوّنات التعبيرية أو من حيث الملفات، و قواعد ضم تلك المُكوّنات و الملفات إلى بعضها البعض و استخدامها سوياً. و نظراً لأنه لم يتم بعد شرح المُكوّنات فإننى أُنَبِّه إلى أن هذا الجزء سيتضح معناه و الأجزاءُ الغامضة فيه أكثر عند المرور على الفصول التى تليه؛ و التى تحوى شروحات مُكوّنات اللغة على نحوٍ تفصيلى. فيُرجَى التنبه إلى هذا.

تكوين البرنامج من المِلفّات files و المُجَلّدات folders:

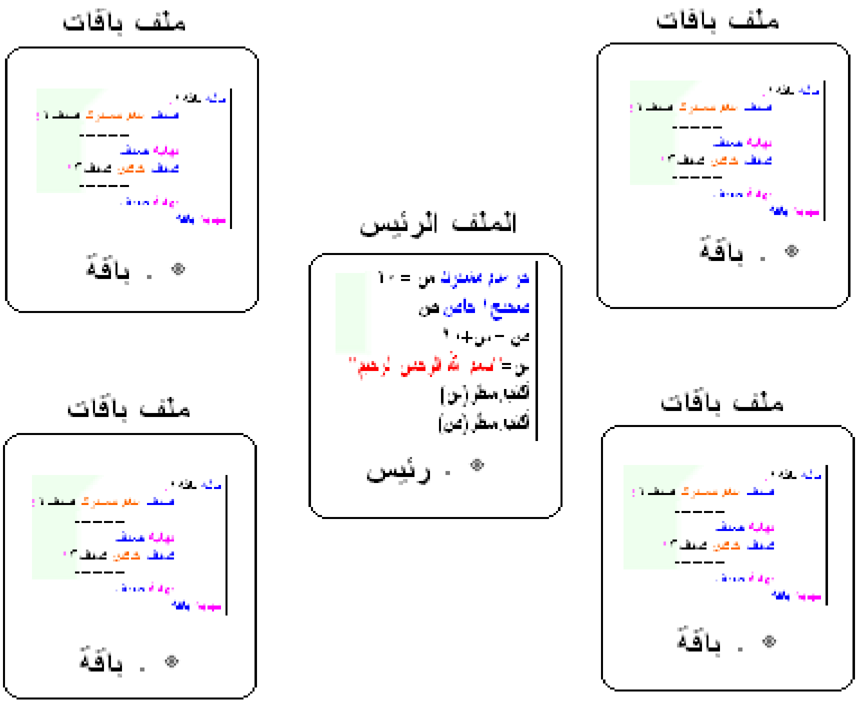
يتكون من ملفاتٍ نصيةٍ (ملفٍ واحدٍ أو أكثر) تُكتب بصيغة اليونيكود (utf-8)، و ملفات البرنامج لها أحد النوعين:

- **الملف الرئيس:**

و هو الملف الذي يحتوى على المسار الرئيس لسير البرنامج و الذي يبدأ التنفيذ به، و امتداد الملف الرئيس هو (*. رئيس) و لا يُمكِن للمشروع الواحد أن يحتوي على أكثر من ملفٍ رئيسٍ واحد.

- **ملف باقات:**

و هو الملف الذي يحتوى على كود صِنْفٍ واحدٍ (كما سيلي توضيحه في الجزء الخاص بالأصناف)، و يُمكِن للبرنامج أن يحتوى على أي عددٍ من ملفات الباقات، و له الامتداد (* . باقة).



و يُمكِن لملفات الباقات أن تُوضَع:

- داخل مُجَلَّد البرنامج (و هو نفس المُجَلَّد الذي يُوجَد به الملف الرئيس)، أو:
- داخل مُجَلَّداتٍ أخري (سواءٌ أكانت تلك المُجَلَّدات الأخري موجودةً في مُجَلَّد البرنامج أم لا)،

و تُسمَّي هذه المُجَلَّدات الفرعية بالـ(باقات)،

و يُمكِن للبرنامج أن يحتوي علي أي عددٍ من الباقات.

و إذا كانت هناك باقةٌ نرغب في استخدامها: فيجب ضمها باستخدام أمر أضم الذي سيلي التنويه إليه.

تكوين الملف الرئيس باختصار:

يتكون الملف الرئيس من العناصر التالية:

[أوامر ضم المكتبات(إن وُجِدَت) [
[أوامر البرنامج [
[الإجراءات المنفردة (إن وُجِد أيٌ منها) و/أو تعريفات الألقاب المنفردة (إن وُجِد أيٌ منها) و/أو الأصناف (إن وُجِد أيٌ منها) [

توضيحات:

- الترتيب أمرٌ لازم الإتباع على النحو سابق التوضيح، فالإجراءات و/أو الأصناف و/أو الألقاب لا يُمكِنها أن تسبق أوامر البرنامج، و هكذا فإن أى مُخالَفةٍ للترتيب سيَنتج عنها خطأٌ مُناسبٌ من المُترجِم.
- كل أمرٍ في إبداع ينتهى بانتهاء سطره، و لا يُمكِن أن يمتد الأمر الواحد إلى أكثر من سطرٍ واحدٍ إلا إذا كان الأمر نفسه مُركَّباً (مثل جملة لو الشرطية مثلاً) أو تم استخدام مازِج الأسطر (الذي سيلي ذكره بالتفصيل).
- يتم ضم أي باقةٍ من باقات المكتبة باستخدام الصيغة التالية:

أضم الباقة1_الباقة2_الباقة3

حيث تحتوي الباقة (الباقة1) على الباقة (الباقة2) و التي تحتوى بدورها على الباقة (الباقة3).

و معني أمر الضم أن مُكوِّنات هذه الباقة (من أصنافٍ و ألقاب) يمكن استعمالها بشكلٍ مُباشرٍ و كأنها قد تم تعريفها داخل الملف الرئيس نفسه، بدون الاضطرار إلي كتابة اسم الباقة اولاً ثم يليه اسم المُكوِّن المرغوب في استخدامه في كل مرةٍ نرغب في استخدام مُكوِّنٍ داخل الباقة.

مثال: بدلاً من الجمل الطويلة التالية:

الباقة1_الباقة2_الباقة3_اسم. صنف1

الباقة1_الباقة2_الباقة3_اسم. صنف2

الباقة1_الباقة2_الباقة3_اسم. صنف3

يمكننا كتابة

أضم الباقة1_الباقة2_الباقة3

و بعدها نستخدم الأ صناف الداخلية مبا شرةً كما يلي:

ا سم. صنف1

ا سم. صنف2

ا سم. صنف3

- (أوامر البرنامج) تَحتوى على الأوامر التى سيتم تنفيذها فى البرنامج مثل ا ستدعاءات الإجراءات procedures calls و ا ستنساخات الأ صناف classes instantiatings و/أو الألقاب enumerations و إ شهار المُتَغَيِّرات variables definitions و غيرهن من الأوامر، أى أنها هى المسار الرئيس للبرنامج الذي يُشبه مُحتوَيات الدالة main في لغات عائلة الـC. و يجب في كل برنامج وجو د أمرٍ وا حدٍ علي الأقل في ذلك المسار.
- مُدخَلات البرنامج عند تشغيله من سطرٍ الأوامر تُوجَد فى جدولٍ (سيتم شرح معني الجدول فيما بعد) نوع عنا صره هو نص، و هو جدولٌ ذو بُعدٍ وا حدٍ يُسمى (دخل)، و يتم التعامل معه كأي جدولٍ نصي عا دي.

و يحتوى على المُعامِلات arguments التى تم تمريرها للبرنامج عند ا ستدعائه، بدايةً من العنصر صا حب المُمَيّز index ذي القيمة (1) إلي آخر عنا صره.

- الإجراءات و/أو الأ صناف و/أو الألقاب الموجو دات في الملف الرئيس يُمكِنها أن تَري بعضها البعض بشكلٍ عا دي، و لكن لا يُمكِن لأيها أن يَري المُتغيِّرات و/أو الثوابت و/أو الجداول و/أو كائنات الأ صناف و/أو كائنات الألقاب التي تُوجَد في مسار البرنامج الرئيس.

تكوين ملف الباقات:

يتكون كل ملف باقةٍ من العنا صر التالية:

[أوامر ضم المكتبات(إن وُجدت)	]
[تعريفات الألقاب المنفر دة (إن وُجِد أيٌ منها)	]
[تعريف الصِّنْف	]

توضيحات:

- يجب أن يَحتوى كل ملفٍ على صِنْفٍ واحدٍ(و احدٍ فقط) له نفس ا سم الملف، فإن لم يَحتو على أى صِنْفٍ حصل المُبرمج على خطأٍ نحوي من المُترجم.
- الألقاب المُنفر دة التي يتم تعريفها في ملف الباقة تُعتبَر ضمن مُكوّنات الباقة ذاتها، بحيث متي ما تم ضم الباقة إلي البرنامج (با ستخدام الأمر أضم) فإنه يُمكِن ا ستخدام تلك الألقاب بشكلٍ مُبا شر و كأنه قد تم تعريفها داخل البرنامج.

شروط التسمية فى إبداع:

هذه الشروط يجب أن تتوافر فى الأسماء المُعطاة من قِبَل المُستخدِم، مثل:

- المُتغيِّرات المُختلفة التى يُشْهِرها،
- الإجراءات،
- الألقاب و كائناتها،
- الجداول،
- أسماء ملفات الباقات،
- الأصناف و كائناتها،

و غيرهن من المُكوِّنات المُستحدَثة التى يكتب هو أسماءها بنفسه.

و **الشروط** هى:

- يُمكِن أن يحتوى الإسم على تجميعةٍ من الحروف و الأرقام و النقطة،
- عدم البدء بأرقام أو بنقطة،
- عدم الإحتواء على أي علامةٍ سواءٌ أكانت من العلامات المحجوزة فى اللغة (و هن المكتوبات فى السطر التالي) أو غيرهن

= # × ÷ + - _ }) ({ > < >< <> | : \ / ،

- ألا يكون الإسم من الكلمات المحجوزة reserveds words مثل إجراء و صنف،
- أن يكون الإسم مُتصل الحروف و النقاط و الأرقام، أى لا يحتوى على مساحاتٍ بيضاء مثل المسافات spaces أو الإزاحات indentations.
- ألا يكون مُتكررا داخل حَيِّز الأسماء namespace الخاص به (أى تم استخدامه من قبل مع مُتغيِّرٍ أو إجراءٍ أو صِنْفٍ أو خلافه داخل ذلك الحَيِّز).

مازج الأسطر:

فى إبداع ينتهى سطر الأوامر بالنزول إلى سطرٍ جديد (و ليس كما فى عائلة الـC التى تنتهى الجمل الأمرية فيها بالفاصلة المنقوطة ;). و لكن يُمكِن باستخدام علامة | جعل المُترجِم يَعتبر السطر التالى جزءاً من السطر الحالي حتى يبلغ تلك العلامة، و يُعتبَر ما بعدها تعليق.

مثال1:

```
حساب الحساب.الأول = حساب(50000          |
(20
```

و هذا يُماثل تماماً كتابة:

```
حساب  الحساب.الأول = حساب(50000 20)
```

مثال2:

```
متغير1 = متغير2 + إجراء1(معامل1     | هذا تعليق
                        معامل2      |
                        معامل3)
```

و هذا يُماثِل كتابة:

```
متغير1 = متغير2 + إجراء1(معامل1  معامل2  معامل3)
```

أنواع البيانات فى إبداع:

تنقسم أنواع البيانات فى إبداع إلى:

- أنواع بياناتٍ أ صليةٌ تُسمَّى المُتغيِرات و الثوابت و كائنات الألقاب.
- أنواع بياناتٍ مُرَكَّبةٌ تتألف من تجميعةٍ من النوع السابق، و هى:
  - الجداول.
  - كائنات الأ صناف.

و سيأتى ذكر عنا صر هذين النوعين بالتفصيل فيما يلى.

التعليقات:

الصياغة:

الصياغة	الإ ستخدام	المُكافئ فى عائلة السى
تبدأ بـ \ و تنهى بـ /	لسطرٍ واحدٍ أو لأ سطرٍ متعددة	/* */

قواعد الا ستخدام:

يُمكِن ا ستخدام التعليقات فى أى مكانٍ داخل الكو د.

مثال:

\ هذا مثالٌ لتعليق سطر واحد /

\ هذا مثالٌ
علي
تعليقٍ لعدة أ سطر /

المُتَغَيِّرات و الثوابت

هن ما يُسمي بالـvariables و الـconstants علي الترتيب في اللغات الإنقليزية، و المتغيرات عبارة عن حاوياتٍ لقِيَم (و ليست مؤشراتٍ pointers لقِيَم)، بمعني أنه إذا ساوينا متغيراً ما بمتغيرٍ آخرٍ فإن هذا يعني أننا أسندنا قيمة الثاني إلي الأول بدون ربطٍ بينهما زيادة عن ذلك، و لو تغيرت قيمة الثاني بعد المساواة فإن قيمة الأول لا تتأثر مطلقاً. أما الثوابت فهن متغيراتٌ لا تتغير قيمتها الأولي نهائياً.

الصياغة:

النوع مستوى.الوصول الإستنساخ الثبات المتغيرالأول= قيمة المتغيرالثانى = قيمة

الأنواع:

النوع	الشرح
رقم	و هو نوعٌ عامٌ يصلح لكل الأنواع الرقمية الموجبة و السالبة، و الصحيحة و الكسرية، و يقوم المُترجم compiler أو المُفسر interpreter الخاصين بإبداع بالتكفل باستنتاج مساحة المُتغير الرقمي و ما يحتاجه من الذاكرة تلقائياً.
نص	يُشير إلى سلسلةٍ نصيةٍ تصلح لاحتواء النصوص ذات الحرف الواحد (التي تُدعي char في لغة الـjava) و النصوص عديدة الأحرف.
منطق	يأخذ أحد القيمتين (صحيح) و (خطأ).
حر	يُشير إلى أى نوعٍ من الأنواع الفائتة، و كذلك إلى أى كائنٍ مُسْتَنْسَخٍ من الأصناف أو الألقاب (لم يتم بناؤه في الإصدارة 1.0 من المُفسِر interpreter الخاص باللغة).

مستويَات الوصول:

لو تم تعريف المتغير في صنفٍ يكون لها المعاني التالية:

المستوى	الشرح
عام	يمكن الوصول إليه من أى مكان.
داخلي	يمكن الوصول إليه من داخل صنفه و الأصناف الوارثة له فقط.
خاص (الافتراضي)	يمكن الوصول إليه من داخل صنفه الحاوى له فقط.

الاستنساخ:

الصفة	الشرح

مشترك	**لو تم تعريف المتغير في صنفٍ:** فلا يمكن صنع نسخةٍ منه لكل نسخةٍ من صنفه الحاوى له، بل يتم إنشاء نسخةٍ واحدةٍ لكل نسخ الصنف و تُستخدَم بإسنادها إلى أي كائنٍ من كائنات صنفها. أما **لو تم تعريفه في إجراءٍ:** ففي كل مرةٍ يتم فيها استدعاء ذلك الإجراء يكون ذلك المتغير محتفظاً بالقيمة التي تم إسنادها له في آخر استدعاءٍ سابقٍ لذلك الإجراء. و في كلا الحالتين فإن القيمة التي يتم إعطاؤه إياها في نفس سطر تعريفه تُعطي له في أول مرةٍ فقط، بينما في المرات التالية يكون محتفظاً بآخر قيمةٍ أُسندت إليه كما تم شرحه لكلا النوعين.
بدون صفة (الافتراضى)	**لو كان في صنفٍ:** فسوف تُصنع نسخةٌ منه لكل كائنٍ من النوع الحاوى له و تُعتبر كل نسخةٍ متغيراً حقيقياً مستقلاً فى الذاكرة. أما **لو كان في إجراء** فسيتم إعادته إلي القيمة الافتراضية في كل مرةٍ يتم فيها استدعاء ذلك الإجراء.

الثبات:

الصفة	الشرح
ثابت	قيمته من النوع الثابت أى لا يمكنٍ تغييرها، و هذا النوع يجب إعطاؤه قيمةً ما في نفس سطر تعريفه و إلا اعْتُبر خطأ نحويا.
بدون صفة (الافتراضى)	قيمته من التوع المتغير أى يمكن تغييرها

مثال1:

```
رقم س ص
```

مثال2:

```
رقم ص س = 10
ص = س + 5
أكتب.رقم.سطر(ص)
```

خرج مثال2:

```
15
```

تحويل قيمةٍ منطقيةٍ إلي قيمةٍ نصية:

يُمكن ذلك عن طريق استخدام الإجراء القياسي (منطق.إلي.نص) الذي له الترويسة التالية:

```
إجراء (نص النص) منطق.إلي.نص (منطق المنطق)
```

مثال:

```
أكتب.نص. سطر("القيمة المنطقية هي " + منطق.إلي.نص( صحيح))
```

الإجراءات القيا سية الخا صة بالتعامل مع النصوص:

هناك عددٌ من الإجراءات القيا سية في اللغة للتعامل مع النصوص هي كما يلي:

- الإجراء (حرف.رقم) صاحب الترويسة:

```
إجراء (نص الحرف) حرف.رقم (رقم الرقم  نص النص)
```

و يُستخدم في الحصول علي حرفٍ معينٍ من داخل النص عن طريق رقم ذلك الحرف داخل النص، مع العلم أن الترقيم يبدأ برقم 1 كما في الجداول التي سيلي ذكرها.

- الإجراء (طول.النص) صاحب الترويسة:

```
إجراء (رقم الطول) طول.النص ( نص النص)
```

و يُستخدم للحصول علي طول نصٍ معينٍ (أي عدد الأحرف و المسافات الفارغة فيه).

- الإجراء (اقتباس) صاحب الترويسة:

```
إجراء (نص الاقتباس) اقتباس (رقم البداية  رقم النهاية  نص النص)
```

و يُستخدم في الحصول علي جزءٍ من النص يبدأ بالحرف صاحب الرقم (البداية) و يتضمنه. و ينتهي بالحرف صاحب الرقم (النهاية) و يتضمنه.

مثالٌ علي ا ستخدام تلك الإجراءات:

```
أكتب.نص.سطر("طول النص (بسم الله الرحمن الرحيم)  هو " + إلي.نص(طول.النص("بسم الله الرحمن الرحيم")))
أكتب.نص.سطر("الحرف رقم 2 من (محمد)  هو " + حرف.رقم(2   "محمد"))
أكتب.نص.سطر("الكلمة الثانية في جملة (بسم الله الرحمن الرحيم) هي " + اقتباس( 5   8   "بسم الله الرحمن الرحيم"))
```

و خرج هذا البرنامج:

```
طول النص (بسم الله الرحمن الرحيم)  هو 22
الحرف رقم 2 من (محمد)  هو ح
الكلمة الثانية في جملة (بسم الله الرحمن الرحيم)  هي الله
```

التحويل من و إلي النوع حر:

النوع حر يماثِل النوع var في اللغات الإنقليزية. و يمكن إسنا د أي قيمةٍ إليه مهما كان نوعها. لكن لإرجاع تلك القيمة إلي نوعها الأ صلي فإن هذا يحتاج إلي قواعدٍ خا صة. تلك القواعد لم يتم الانتهاء من تصميمها في إبداع لأنه لم يتم بناء النوع حر فيها حتي الآن (أي في الإ صدارة 1.0).

المُعامِلات المنطقية و الحسابية

المُعامِلات المنطقية:

المُعامِل	الشرح
و	تكافئ AND في التعبير الإنقليزي
أو	تكافئ OR في التعبير الإنقليزي
ليس	تكافئ NOT في التعبير الإنقليزي
=	هل يساوي (إذا جاءت في جملةٍ منطقية)، تُكافيء == في لغات عائلة الـC
#	لا يساوي
>	أكبر من
<	أقل من
>=	أكبر من أو يساوي
<=	أقل من أو يساوي
><	لا يساوي
<>	لا يساوي

المُعامِلات الحسابية:

المُعامِل	الشرح
+	جمع
-	طرح
×	ضرب
÷	قسمة
=	للمساواة
^	أس

الأسبقية في التنفيذ:

1) الأقواس،
2) الأ سس،
3) المعاملات المنطقية >= <= = # >< <> في
4) المعامل المنطقي ليس،
5) المعامل المنطقي و،

6) المعامل المنطقى أو،
7) الضرب و القسمة أيهما أسبق من ناحية اليمين،
8) الجمع والطرح أيهما أسبق من ناحية اليمين.

الإجراءات

نبذةٌ بسيطة:

الإجراء هو كتلةٌ من الأوامر تُكتَب كوحدةٍ واحدةٍ و يُطلَق عليها اسمٌ مُعيَّنٌ يتم استدعاؤها به لتُنفَّذ في كل مرةٍ نَود تنفيذها، و يُمكننا تمرير مُعَامِلاتٍ مُعينة النوع إلى الإجراء لكي يعمل عليها، و يُمكِننا الحصول على أكثر من مُخرَج من الإجراء الواحد.

و هو يُماثِل الـfunction في اللغات الأخري مع بعض الاختلافات.

الصياغة:

إجراء النوع مستوى.الوصول الاستنساخ التغطية (المُخرَجات) اسم.الإجراء(المُعامِلات): \ جسم الإجراء /

النوع:

الصفة	الشرح
ثابت (الافتراضي)	هو الإجراء العادي المعروف في باقي لغات البرمجة و الذي يُنَفِّذ الكود المكتوب فيه فقط عند استدعائه بشكلٍ صحيح.
حر	هو الإجراء الذي يقبل وضع استدعاءاتٍ لإجراءاتٍ أخرى في آخر كوده في زمن التنفيذ، بحيث تُنَفَّذ بعد الكود الثابت المكتوب قبل زمن التنفيذ، و يقبل كذلك حذف الاستدعاءات التي وُضِعت فيه أثناء زمن التنفيذ. و يتفق مع الإجراء الثابت في كل صفاته بينما يمتلك خصائص أخرى لا تشاركه فيها الإجراءات الثابتة، و سيأتي شرح تلك الصفات فيما بعد. يُشبِه هذا النوعُ الـ delegations في الفكرة الأساسية.
مخصص	هو الإجراء الذي لا يحمل إلا كودا مكتوبا بلغة تجميع assembly code خاصة بمعالج معين يتم تحديد اسمه بالطريقة التي سيأتي شرحها في بند (الإجراءات المخصصة)، و الإجراء المخصص هو إجراء ثابت و لكنه لا يمكن أن يحتوي على كود مكتوب بإبداع، بل يجب أن يكون كل الكود مكتوب بلغة التجميع (لم يتم بناؤه بعد في الإصدارة رقم 1.0 من مُفسِّر اللغة).

مستوَيات الو صول:

إذا كان الإجراء في صنفٍ فيكون معناها:

المستوى	الشرح
عام (الافتراضي)	يمكن الو صول إلي الإجراء من أى مكان.
داخلي	يمكن الو صول إليه من داخل صنفه و الأ صناف الوارثة له فقط.
خاص	يمكن الو صول إليه من داخل صنفه فقط.

الا ستنساخ:

إذا كان الإجراء في صنفٍ فيكون معناها:

الصفة	الشرح
مجرد	ليس في الإجراء كود: بل مجرد الترويسة التي فيها و صفٌ للإجراء و خوا صه فقط.
مشترك	لا تُصنع منه نُسخةٌ لكل كائنٍ مُستنسَخ، و لا يجوز داخله استخدام أي مكونٍ من مكونات الصنف الغير مشتركة.
بدون صفة (الافتراضية)	هو الإجراء المُستنسَخ العادي.

التغطية:

إذا كان الإجراء في صنفٍ فيكون معناها:

الصفة	الشرح
سقف	لا يمكن تغطيته فى الأ صناف التى ترث صنفه.
بدون صفة (الافتراضية)	هو الإجراء العادي الذى يقبل التغطية.

مثال 1:

إجراء عام مشترك (رقم س منطق ص) ماعددالحالات(نص الحرف نص المحتوى):
\ الكود /

مثال 2:

إجراء عام مجرد (رقم أ) أوجدالجذرالتربيعى(رقم س)

و مهما كان عدد مُدخَلات أو مُخرَجات الإجراء فيجب فى كل الأحوال إحاطتهما بقوسين و إلا حصل المبرمج على خطأٍ نحوى. أما إذا كان الإجراء لا يُعيد أى مُخرَجٍ أو لا يأخذ أى مُدخَلٍ فلا يمكننا أن نكتب

كلاً من و أياً من القوسين الخا صين بكل حالةٍ فارغين بل يجب ألا نكتبهما على الإطلاق، أى أن الأ شكال التالية مسموحٌ بها جميعها فى إبداع:

الشكل الأول:

```
إجراء عام مشترك إجراءما:
        \ الكود /
```

الشكل الثانى:

```
إجراء عام مشترك (رقم  أ) إجراءما:
        \ الكود /
```

الشكل الثالث:

```
إجراء عام مشترك  إجراءما(رقم  س):
        \ الكود /
```

بينما الأ شكال التالية خاطئةٌ و لا يُسمح بها فى إبداع، و تؤدى للحصول على خطأٍ نحوىٍ من المترجم:

الشكل الأول:

```
إجراء عام مشترك () إجراءما():
        \ الكود /
```

الشكل الثانى:

```
إجراء عام مشترك () إجراءما:
        \ الكود /
```

الشكل الثالث:

```
إجراء عام مشترك  إجراءما():
        \ الكود /
```

المُدخلات و المُخرَجات:

يُمكِن للإجراء فى إبداع أن يُعِيد أكثر من قيمة، فإذا ما أخذنا الإجراء (ماعددالحالات) كمثالٍ لنا فسنجد أننا سنستخدم المتغيرات المُشْهَرة كمُخرَجاتٍ (فى هذه الحالة س، ص) بصورةٍ عاديةٍ فى الإجراء، و آخر قيمٍ ستكون مُخزنةً فيهما ستكون هى ما سيتم إعادته عند نهاية تنفيذ استدعاء الإجراء.

و يتم استدعاء الإجراء كما فى المثال التالي:

```
رقم عددالحالات
منطق يمثل.أغلبية
عددالحالات، يمثل.أغلبية = ماعددالحالات("م"   "بسم الله الرحمن الرحيم")
```

حيث نفترض أن الإجراء (ماعدالحالات) هو نفسه فى المثال1، و بالتالى نرى أن شروط استدعاء الإجراء متعدد الخرج فى حالة عدم الرغبة فى إهمال خرج ما أو أكثر من مُخرَجات الإجراء هى:

1. وجود متغيراتٍ لها نفس نوع و ترتيب مُخرَجات الإجراء فى الجانب الأيمن من أمر استدعاء الإجراء.
2. الفصل بين كل متغيرين فى الجانب الأيمن من الاستدعاء بفاصلة (،).

و عدم توافر شرطٍ من الشروط السابقة فى استدعاء الإجراء متعدد الخرج يؤدى إلى ظهور الخطأ البرمجي المناسب من المصحح اللغوي.

أما فى حالة الرغبة فى إهمال خرج ما أو أكثر من مُخرَجات الإجراء فهو أمرٌ له شروطٌ معينةٌ و سيأتى ذكره بعد قليلٍ فى فصل (إهمال خرج الإجراء عند استدعائه).

تمرير الجداول كمُعامِلاتٍ للإجراءات و مُخرَجاتٍ منها:

1. الجداول كمُعامِلاتٍ للإجراءات:

إذا ما رغبنا فى تمرير جدولٍ للإجراء كمُعامِلٍ له فنستخدم الصيغة التالية فى مُعامِلات الإجراء:

```
النوع{عددالأبعاد} الاسم
```

مثال:

```
إجراء (رقم س) ماعدالحالات(نص س نص{1} ص):
\ الأوامر /
```

2. الجداول كمُخرَجاتٍ من الإجراءات:

إذا ما رغبنا فى تعريف جدولٍ للإجراء كمُخرَجٍ منه فنستخدم الصيغة التالية فى مُخرَجات الإجراء:

```
النوع{عددالأبعاد} الاسم
```

و هذه هى تقريباً نفس الصياغة التى تُستخدَم فى الأصل لتعريف الجداول لكن بدون وجود صفاتٍ مُعيّنَةٍ هى: مستوَيات الوصول و الاستنساخ، و التى سيتم شرحها فيما بعد عند التعرض للجداول نفسها.

تحميل الإجراءات overloading:

هو إمكانية تعريف أكثر من إجراءٍ فى حيز أسماءٍ واحدٍ (مثلاً: نفس الملف الرئيس أو نفس الصنف الحاوي) لهم نفس الاسم، و لكن يجب أن يكون هناك اختلافٌ بين الإجراءات المُحَمَّلة في عدد و/أو أنواع و/أو ترتيب أنواع قيم المُدخَلات.

فإذا لم يكن هناك أى اختلافٍ بين الإجراءات المُحَمَّلة يندرج تحت الاختلافات السابقة الذِكر فسيُعد وجود الإجراءات المُحَمّلة فى حيز تسميةٍ واحدِ خطأ من قِبَل مترجم اللغة.

أمثلةٌ على تحميلاتٍ صحيحةٍ لإجراءات:

```
إجراء (رقم س منطق ص) إجراءما(رقم س):
    \ جسم الإجراء /
إجراء (رقم س منطق ص) إجراءما(نص س):
    \ جسم الإجراء /
```

إهمال خرج الإجراء عند استدعائه:

إذا أردنا استدعاء إجراءٍ يُعيد قيمةً أو أكثر و لكن بدون استخدام بعض أو كل تلك القيم (باستقبال قيمتها فى متغيرات) فيُمكننا ذلك بالأشكال التالية:

1. الاستغناء عن كل المُخرَجات:

و فى هذه الحالة يمكن استدعاء الإجراء بكتابة اسمه مباشرةً ثم تليه قائمة المُعامِلات المُرسَلة إلي الإجراء.

مثال:الإجراء (إجراءٌ.ما) له تحميلٌ واحد:

```
إجراء.ما(50)        \ تم إهمال جميع مُخرَجات الإجراء /
إجراء.ما("نص")

إجراء (منطق ص) إجراء.ما(رقم  س):
    \ جسم الإجراء /
إجراء (رقم ع نطق ص) إجراء.ما(نص س):
    \ جسم الإجراء /
```

2. الاستغناء عن بعض المُخرَجات:

يكون ذلك بعدم كتابة أى شيءٍ في مكان الخَرج المرغوب في إهماله فى جملة استدعاء الإجراء. و يُمكننا تكرار هذا مع أى مُخرَجٍ نرغب فى إهماله حتى لو كررناها مع جميع مُخرَجات الإجراء.

مثال:

```
، ع = إجراء.ما(5)     \ تم إهمال الخرج الأول فقط للإجراء /
، = إجراء.ما(50.0)    \ تم إهمال جميع مُخرَجات الإجراء /
```

```
إجراء (منطق ص رقم ك) إجراء.ما(رقم س):
\ جسم الإجراء /
إجراء (منطق ص رقم ك) إجراء.ما(نص س):
\ جسم الإجراء /
```

إسناد قيم ابتدائية لمدخلات و مخرجات الإجراء:

فى حالة ما إذا أردنا أن نُعطى المتغيرات التى يُمرّرها المستخدم إلى الإجراء (أي المُدخَلات) أو المتغيرات الناتجة عنه (أي المُخرَجات) قيمةً ابتدائيةً معينةً بحيث تُعطى لها فعلياً إذا:

- لم يُمرّر المستخدم للإجراء قيمةً مكافئةً لذلك المُدخَل،
- أو لم يتم إعطاء قيمةٍ للمُخرَج في داخل الإجراء في زمن التنفيذ.

يمكننا أن نفعل ذلك باتباع الصيغة التالية:

```
إجراء النوع مستوى.الوصول الإستنساخ التغطية (خرج1 = قيمة1 ....) اسم.الإجراء(معامل1 = قيمة1 ....):
\ جسم الإجراء /
```

بالنسبة للمُدخَلات:

إذا مَرّر المستخدم للإجراء قيمةً للمُعامِل الأول فقط: يتم اعتبار قيمة المُعامِل الثاني كما وُضِعت فى القيمة الافتراضية، و إذا لم يُمرّر المستخدم أى قيم للمُعامِلات: يتم أخذ كل القيم الافتراضية و إسنادها للمُعامِلات المقترنة بها، و يصح استخدام خاصّية التحميل مع الإجراءات التى لمُعامِلاتها قيمٌ افتراضيةٌ (و إن كان هذا قد يؤدي إلي ارتباك المبرمج إذا لم يُحسِن التعامل معه جيداً).

و لكن يجب مراعاة الأمور التالية:

- أن يكون المُعامِل/ات المهمل/ة فى تمرير القيمة فى آخر قائمة المعاملات، فلا يُمكن إهمال تمرير المُعامِل الأول فى قائمة المُعامِلات و تمرير الثانى لأن هذا سيجعل المترجم يضع القيمة المُمرّرة للمُعامِل الثانى فى المُعامِل الأول و هو ما سيتسبب فى خطأٍ نحويٍ إذا تضاربت أنواع المعاملات، بالإضافة إلي كونه غلطا منطقيا فى البرنامج.
- إذا ما أسمينا توقيعَ الإجراء بعد استبعاد المُعامِلات التي لها قيمٌ ابتدائيةٌ بـ(التوقيع الأدني): فيجب ألا يتشابه التوقيع الأدني لكل إجراءٍ مع التوقيع الأدني لأي إجراءٍ آخر له نفس الاسم.

  مثالٌ علي التوقيع الأدني:

  الإجراء:

```
إجراء الإجراء (رقم س  نص ص = "بسم الله"):
\ الكود /
```

توقيعه الكامل: ***رقم نص***

توقيعه الأدني: ***رقم***

و بالنسبة للمُخرَجات:

فإذا قام المُبرمِج بإسناد قيمةٍ للخرج داخل الإجراء و تم تنفيذ هذا الإسناد في زمن التشغيل فلن يتم إستخدام القيمة الافتراضية حينها. أما إذا لم يفعل: فسيتم استخدام القيمة الافتراضية.

و لا يُشترَط هنا أن تكون المخرجات ذات القيم الافتراضية في آخر القائمة، بل يُمكن أن تكون في أي مكان.

مثال: الإجراء:

إجراء (رقم س منطق ص) ماعد دالحالات(رقم الرقم نص الحرف = "ل" نص المحتوى= "الله أكبر"):
\ جسم الإجراء /

يمكن استخدامه كما يلي:

الإستخدام	قيم المتغيرين (الحرف) و (المحتوى)
ماعد دالحالات(1)	الحرف = 'ل' المحتوى = "الله أكبر"
ماعد دالحالات(2 "ق")	الحرف = 'ق' المحتوى = "الله أكبر"
ماعد دالحالات(3 "ق" "بسم الله الرحمن الرحيم")	الحرف = 'ق' المحتوى = "بسم الله الرحمن الرحيم"

بينما يُسبب الاستخدام التالي خطأً نحوياً:

ماعد دالحالات("بسم الله الرحمن الرحيم")

لأنه تم تمرير متغيرٍ من النوع نص فقط إلى الإجراء (ماعد دالحالات) الذى أول معاملٍ له من النوع رقم و بالتالى فإن النوعين لا يتفقان.

كسر تنفيذ أوامر الإجراء:

في أحيانٍ معينةٍ قد نرغب في إيقاف تنفيذ إجراءٍ معينٍ عند نقطةٍ محددة، و في هذه الحالة يمكننا استخدام الأمر أنهي() أو الأمر أتخطى() لأداء هذه المهمة.

مثال:

```
إجراء تجربة.أنهي :
    لو صحيح :
        أنهي(تجربة.أنهي)
    أكتب.نص.سطر("هذا أمر لن يتم تنفيذه")
```

و لأن أنهي() لا فائدة منطقية له إن وُضِع في صلب الإجراء مباشرةً: فقد تم منع هذا بحيث لو فعله أحدهم لكان خطأ. و لا يصح وضعه إلا في داخل جملة (لو) أو ما شابهها.

كما أنه يمكننا معاملة البرنامج كله و إيقاف تنفيذه عن طريق استخدام الكلمة المفتاحية البرنامج بين قوسي الأمرين أنهي و أتخطي.

مثال:

```
لو صحيح :
    أنهي(البرنامج)
أكتب.نص.سطر("هذا أمر لن يتم تنفيذه")
```

و يجب التنبه أنه لا يمكن كتابة أية أكوادٍ بعد أتخطي و أنهي في نفس كتلتهما التكويدية code block و إلا اعتُبِر هذا خطأ.

مثال:

```
لو صحيح :
    أنهي(البرنامج)
    أكتب.نص.سطر("هذا خطأ")        ×
أكتب.نص.سطر("هذا أمر لن يتم تنفيذه")
```

طلب معالجة غلطٍ من داخل الإجراء:

يُمكن في صلب الإجراء إلقاء غلطٍ exception throwing بالطريقة التي سيتم شرحها فيما بعد عند الحديث عن معالجة الأغلاط.

مثال:

```
إجراء تجربة.معالجة.غلط :
    أعالج(غلط())
```

ما ينفرد به الإجراء الحر و لا يُشاركه فيه الإجراء الثابت:

1. إضافة إجراءٍ إلى قائمة استدعاءات الإجراء الحر في زمن التنفيذ:

استخدام الصياغات التالية:

الصيغة 1:

```
في اسم.الإجراءالحر(التوقيع) أضيف(استدعاء الإجراء)
```

و يجب مراعاة القواعد التالية:

1. بعد اسم الإجراء الحر يمكن إهمال كتابة الأقواس و التوقيع إن لم تكن هناك تحميلاتٌ للإجراء الحر، فإن كانت هناك تحميلاتٌ وجب كتابة أنواع المتغيرات بالترتيب الصحيح بدون وضع أسماءٍ لها (و هذا هو المقصود بالتوقيع).

مثال:

```
في نداءالحالات أضيف( ماعدادالحالات("ق"))
```

مثال:

```
في نداءالحالات(رقم نص) أضيف( ماعدادالحالات("ق"))
```

2. بعد اسم الإجراء المُضاف يمكن إهمال كتابة القوسين و المُعامِلات إن كان له نفس توقيع الإجراء الحر، و ساعتها ستكون قيم المُعامِلات المُمَرَّرة له هي نفسها قيم المُعامِلات المُمَرَّرة للإجراء الحر الذي تم ضمه إليه.

3. بخصوص القيم التي تُوضع كمُعامِلاتٍ في أمر استدعاء الإجراء: سوف يتم حسابها مرةً واحدةً عند تنفيذ أمر الإضافة، و في كل مرةٍ يتم فيها تنفيذ الإجراء الحر الذي تتم الإضافة له سيتم استدعاء الإجراء المُضاف اعتماداً علي قيمة المُعامِلات التي تم حسابها عند الإضافة.

الصيغة2:

```
في اسم.الإجراءالحر1(التوقيع) اسم.الإجراءالحر2(التوقيع) ... أضيف(استدعاء الإجراء)
```

و النقاط الثلاث معناها أسماء إجراءاتٍ أخري لو أردنا كتابة المزيد منها، و هنا يجب مراعاة كل شروط الصياغة1، كما أنه عند إهمال كتابة توقيع الإجراء المُضاف فيجب أن يكون توقيع كل الإجراءات الحرة واحداً و أن يتفق توقيع الإجراء المُضاف مع ذلك التوقيع.

2. حذف إجراءٍ من قائمة استدعاءات الإجراء الحر فى زمن التنفيذ:

استخدام الصياغات التالية:

الصيغة1:

```
في اسم.الإجراءالحر1(التوقيع) ... أحذف.آخر.نداء(اسم.الإجراء(التوقيع))
```

حيث في هذه الصياغة سيتم حذف آخر نداءٍ للإجراء الموجود بين قوسي الأمر أحذف.آخر.نداء في الإجراءات الحرة المذكورة، و يجب مراعاة نفس القواعد التى نبهنا إليها فى صياغات الإضافة.

و إذا لم تكن هناك أي نداءاتٍ للإجراء المُراد حذف نداءاته من الإجراءات الحرة: فلن يتسبب ذلك في أي خطأٍ في زمن التنفيذ.

الصيغة 2:

في اسم.الإجراءالحر1(التوقيع) ... أحذف.كل.نداء(اسم.الإجراء(التوقيع))

حيث في هذه الصياغة سيتم حذف كل نداءٍ للإجراء الموجود بين قوسي الأمر أحذف.كل.نداء في الإجرءات الحرة المذكورة، و يجب مراعاة نفس القواعد التى نبهنا إليها فى صياغات الإضافة.

و هنا أيضاً إذا لم تكن هناك أي نداءاتٍ للإجراء المُراد حذف نداءاته من الإجراءات الحرة: فلن يتسبب ذلك في أي خطأٍ في زمن التنفيذ.

4. الجمع بين الإضافة و الحذف:

يمكن الجمع بين العديد من الإضافات لاستدعاءات إجراءاتٍ إلى الإجراء الحر و/أو الحذف منها فى تعبيرٍ واحد، كما يلي:

```
فى اسم.الإجراءالحر1(التوقيع) ...:
    أضيف(استدعاء الإجراء)
    أحذف.آخر.نداء(استدعاء الإجراء)
    ....
```

مثال:

```
فى نداءالحالات:
    أضيف(ماعدالحالات("م"))
    أحذف.كل.نداء(ماعدالحالات(نص))
```

5. التفريغ الكامل:

بعد تنفيذ الإجراء الحر باستدعائه لن يتم تلقائياً حذف كل الإجراءات التى أُضِيفت إليه فى زمن التنفيذ بشكلٍ ذاتى، بل إذا أراد المبرمج حدوث هذا فيجب أن يأمر بذلك صراحةً عن طريق استخدام الأمر (أحذف.الكل) كما يلى:

الصياغة 1:

في اسم.الإجراءالحر(التوقيع) ... أحذف.الكل()

و هنا أيضاً كما فى الإضافة و الحذف يمكن إهمال كتابة التوقيع و الأقواس بعد اسم الإجراء الحر إن لم تكن هناك تحميلاتٌ له، فإن كانت هناك تحميلاتٌ وجب كتابة توقيعه.

مثال1:

```
في نداءالحالات() أحذف.الكل()
```

و بفرض وجود تحميلٍ للإجراء(نداءالحالات) نكتب:

```
في نداءالحالات(رقم نص) أحذف.الكل()
```

الصياغة2:

```
في اسم.الإجراءالحر(التوقيع) ... :
    أحذف.الكل()
```

ما ينفرد به الإجراء المُخصَّص و لا تشارِكه فيه الأنواع الأخرى:

(لم يتم بناء الإجراء المُخصَّص في الإصدارة رقم 1.0 من مُفسِّر اللغة).

الإجراءات المُخَصَّصة هي إجراءاتٌ يرتبط كل واحدٍ منها بمُعالِجٍ مُعَيَّن، بحيث أنه:

- تكون الصياغة:

```
إجراء مخصص مستوى.الوصول الإستنساخ التغطية (المُخرَجات) اسم.الإجراء(المُعامِلات):
يخص نوع.المعالج :
    \ جسم الإجراء /
```

حيث أن كلمة يخص هي كلمةٌ مفتاحيةٌ يجب كتابتها لتنبيه المُترجِم إلى أن هذا الإجراء مُخصَّصٌ للمُعالِج التالي إسمه. أما نوع المُعالِج الذى تتم كتابته بعد كلمة يخص فهو عنصرٌ من اللقب القِياسِي (بيئة) الذي يُعبِّر كل عنصرٍ من عناصره عن نوعٍ مُعيَّنٍ من المُعالِجات، أو عن عائلةٍ من المُعالِجات مُتوافِقةٌ مع بعضها البعض.

مثالٌ تخيلي:

```
إجراء مخصص إجراءما:
يخص بيئة_إنتل8086 :
    \ جسم الإجراء /
```

- داخل كل إجراءٍ مُخصَّصٍ يجب كتابة الأوامر بلُغة التجميع assembly language الخاصة بالمُعالِج الذي تم تخصيص ذلك الإجراء له بصورةٍ صرفة. و لا يُسمَح بأوامر عالية المُستوَى، ما عدا ما سَيَلي ذِكره. و بالطبع فإن لغات التجميع التي سيتم استخدامها ستكون لغات تجميعٍ عربيةٍ بإذن الله عز و جل.

- في داخل الإجراء المُخصَّص لا يُمكِن استعمال أكوادٍ بلغة إبداع إلا استدعاءات الإجراءات، و التي تُكتَب بنفس الشكل الذي تُكتَب به في برامج إبداع، بشرط ألا يكون هناك إسناد قيمٍ عند الاستدعاء:

مثالٌ تخيليٌ صحيح:

```
أضع 3، م2
إجراءما()
أجمع م1، م2
```

مثالٌ تخيليٌ به خطأ:

```
أضع 3، م2
س = إجراءما()
أجمع م1، م2
```

- إذا كانت هناك تحميلاتٌ للإجراء المُخصّص: فعند الإستدعاء سيتم تنفيذ الإجراء الذي يكون نوع المُعالِج الذي خُصِّصَ له ذلك الإجراء هو نفس نوع المُعالِج الذي يعمل (في حالة المُفسِّر) أو سيعمل (في حالة المُترجِم) عليه نظامُ التشغيل.

أي بفرض أن لدينا الإجراءات التالية:

```
إجراء مخصص إجراء.ما:
يخص بيئة_إنتل8086 :
\ جسم الإجراء /

إجراء مخصص إجراء.ما:
يخص بيئة_ARM :
\ جسم الإجراء /
```

فعند الاستدعاء:

```
إجراءما()
```

و كان المُعالِج الذي يعمل عليه هذا البرنامج هو إنتل8086 فإنه سيتم تنفيذ الإجراء الأول.

و يتضح من المثال السابق أنه يُمكِن لمجموعةٍ من الإجراءات المُخصَّصة أن تمتلك نفس الخصائص من: الاسم، و عدد و أنواع و ترتيب المُدخَلات و المُخرَجات، و مُعامِلات الوصول، و باقي الخصائص الخاصة بإجراءاتٍ أخرى ثابتةٍ غير مُخصَّصة، أو مُخصَّصةٍ لمُعالِجاتٍ أخرى تختلف عن تلك التي خُصِّصت هي لها؛ حيث أن نوع المُعالِج يُعتبَر في حد ذاته مُعامِل تفرقةٍ بين الإجراءات المُحمَّلة.

- عند استدعاء الإجراء المُخصَّص، و في حالة لم يكن هناك تحميلٌ خاصٌ بالمُعالِج الذي يعمل عليه نظام التشغيل الحالي: فسيَنتج عن هذا غلطُ زمن تنفيذٍ مُناسِب.

- إذا كان هناك إجراءٌ ثابتٌ أو حرٌ له تحميلٌ واحدٌ أو أكثر من إجراءاتٍ مُخصَّصةٍ: فعند استدعاء ذلك الإجراء فستكون أولوية التنفيذ للإجراءات المُخصَّصة، أما إذا لم يُوجَد إجراءٌ مُخصَّصٌ خاصٌ بالمُعالِج الحالي فإنه يتم تنفيذ التحميل الثابت أو الحر.

أي بفرض أن لدينا الإجراءات التالية:

```
إجراء إجراء.ما:
    \ جسم الإجراء /

إجراء مخصص إجراء.ما:
    يخص بيئة_ARM :
    \ جسم الإجراء /
```

فعند الاستدعاء:

```
إجراءما()
```

و كان المُعالِج الذي يعمل عليه هذا البرنامج هو ARM: فإنه سيتم تنفيذ الإجراء الثاني؛ لأنه مُخصَّصٌ لهذا النوع من المُعالِجات، مع أن الإجراء الأول يصلح بالطبع للتنفيذ في كل الأحوال.

- في داخل الإجراء المُخصَّص إذا كان هناك استدعاءٌ لإجراءٍ آخرٍ فهنا حالتان:
    - الإجراء المُستدعَى هو إجراءٌ مُخصَّص:

سيتم التعامل مع المُخزّنات registers كأنها مُتغيراتٌ مُشترَكة، أي أن قِيَمها قبل الاستدعاء و عند بداية تنفيذ الإجراء المُستدعَى ستكون واحدة، و قِيَمها في نهاية تنفيذ الإجراء المُستدعَى و بعد انتهاء تنفيذ الاستدعاء و العودة إلي الإجراء المُستدعِي ستكون

واحدة.

- الإجراء المُستدعَي هو إجراءٌ مُخصَّص:

بعد انتهاء تنفيذ الإجراء المُستدعَي سيتم إعادة قِيَم المُخزّنات إلي ما كانت عليه قبل تنفيذ أمر الإستدعاء.

- يُمكِن لهذه الإجراءات أن يكون لها خرجٌ أو دخلٌ من أي نوعٍ من الأنواع البسيطة أو المُركّبة (أي: رقم و/أو نص و/أو منطق و/أو الجداول و/أو كائنات الأصناف و/أو الألقاب). و لكن التعامل داخل الكود لن يكون مُمكناً إلا مع النوعين البسيطين التاليين: رقم و نص. مع مُراعاة قواعد التعامل التي سَيَرِد ذِكْرها في بند (كيفية التعامل مع المُدخَلات و المُخرَجات).

كيفية التعامل مع المُدخلات و المُخرَجات:

نظراً لأن الكود سيكون مكتوباً بلغة التجميع فإن التعامل مع المُتغيّرات (سواءٌ أكانت مُعاملاتٍ مُمرَّرةً أم مُخرَجاتٍ) سيكون با ستخدام عناوينها، و يكون ا ستخلاص عنوان المتغير با ستخدام الإجراء القِيا سِي مرجع كما يلي:

مرجع(ا سم.المتغير)

مثال:الأمر الوهمى التالى:

أحمل [مرجع(س)]، م1

يُفترَض أن معناه تحميل مُحتوَيات العنوان الذى يُشِير إليه عنوان المتغير س (أى قيمة المتغير س نفسه) إلى المُخزّن المُسمّى م1.

مثال2:الأمر الوهمى التالى:

أخزن م2، [مرجع(ص)]

يضع قيمة المتغير ص فى المكان فى الذاكرة الذى يُشِير إليه عنوان المتغير ص، أى فى المتغير ص نفسه.

مثالٌ على إجراءٍ مُخصّصٍ كاملٍ: (كالعادة: أوامر لغة التجميع المكتوبة فى هذه الإجراء أوامرٌ وهميةٌ للتمثيل فقط):

```
إجراء مخصص (رقم ص) إجراءما(رقم س):
يخص بيئة_إنتل8086 :
    أحمل [مرجع(س)]، م1
    أضع 3، م2
    أجمع م1، م2
    أخزن م2، [مرجع(ص)]
```

الأ صْنَاف

نبذةٌ بسيطة:

الصِّنْف هو مُكوّنٌ يُمثِّل تجميعةً من: المتغيرات مختلفة الأنواع و/أو الإجراءات و/أو الأ صناف الأخرى. حيث أنه يُشبه حاوية للإجراءات بحيث تعمل على المتغيرات الموجودة في داخله بجانبها، و هو بذلك يُشبه الـclasses في لغات الـC++ و الـjava و الـC# في الفكرة الأ ساسية و يختلف في الصياغات و التفا صيل، و سيلى شرح كل ذلك بطريقةٍ شاملة.

الصياغة:

صنف مستوى.الو صول الوراثة الاسم يرث صنف1 صنف2 صنف3 .... :
\ تعبير (ما عدا) إن أردنا استعماله /
\ تعبير (تلقيب) إن أردنا استعماله /
\ تعريف حاويات الصنف (المتغيرات و/أو الثوابت و/أو الجداول و/أو كائنات الأ صناف و/أو كائنات الألقاب /
\ تعريف الألقاب الداخلية و/أو الإجراءات الداخلية و/أو الأ صاف الداخلية /

مستوَيات الو صول:

إذا كان الصنف موجوداً في صنفٍ آخرٍ أو في ملف باقةٍ فتكون لها المعاني التالية:

المستوى	الشرح
عام (الافتراضي)	إذا كان **صنفاً في باقةٍ** فيمكن الو صول إليه من أى مكانٍ خارج باقته. أما لو كان **صنفاً داخل صنفٍ آخرٍ** فمعني كونه عاماً هو أنه يمكن الو صول إليه من داخل صنفه الحاوي له و الأ صناف التي ترثه.
داخلي	إذا كان **صنفاً في باقة:** فيمكن الو صول إليه من داخل باقته فقط. أما لو كان **صنفاً داخل صنفٍ آخرٍ:** فيمكن الو صول إليه من داخل صنفه الحاوي له و الأ صناف الوارثة له فقط، أي أنه في هذه الحالة فإن و صف عام يشبه و صف داخلي تماماً في العمل لأنه لا يُمكن الو صول لصنفٍ داخل صنفٍ آخر من أي مكانٍ خارج صنفه الحاوي له.
خاص	إذا كان **صنفاً في باقة:** فلا يُمكن الو صول إليه من خارج باقته، أي أنه في هذه الحالة يتساوي و صفي داخلي و خاص.

	أما لو كان **صنفاً داخل صنفٍ آخرٍ:** فيمكن الوصول إليه من داخل صنفه الحاوي له فقط.

صفات الوراثة:

الصفة	الشرح
سقف	لا يمكن وراثته و يمكن فقط صنع كائناتٍ منه، و في هذا النوع من الأصناف لا يمكن تعريف إجراءاتٍ و/أو أصنافٍ داخليةٍ مجردة لأنه لن تكون هناك فرصةٌ لبنائها في صنفٍ وارث.
مجرد	لا يمكن صنع نسخةٍ منه بل يُمكن وراثته فقط، و عندها يجب بناء كل إجراءاته المجردة.
بدون صفة (الإفتراضي)	الصنف العادي الذي يمكن وراثته أو استنساخه.

مثال:

صنف عام مجرد حساب: \ جسم الصنف /

صِيَغ الاستخدام:

الصيغة1:

اسم.الصنف مستوى.الوصول الاستنساخ ثبات.المرجع اسم.النسخة1 = اسم.الصنف(المعاملات إن وُجِدَت) اسم.النسخة2 = اسم.الصنف(المعاملات إن وُجِدَت) ....

مستويات الوصول:

هي نفسها الخاصة بمتغيرات الأصناف إذا أُشهر الكائن في صنفٍ ما.

الاستنساخ:

هي نفسها الخاصة بمتغيرات الأصناف إذا أُشهر الكائن في صنفٍ ما.

ثبات المرجع:

الصفة	الشرح
ثابت	قيمة المؤشر من النوع الثابت و هو مخصصٌ للنسخة التي يُربَط بها في

	أول مرةٍ (سواء أكان الربط فى جملة الإشهار أم بعدها). أى لا يمكن تغييرها و جعل المؤشر يشير إلى نسخةٍ أخرى من نفس الصنف. و فعل ذلك يؤدى إلى إعلان المترجم حدوث خطأ نحوى، و لو تم استخدام هذه الكلمة فيجب أن يتم حجز مكان في الذاكرة للكائن في نفس الجملة التعريفية.
بدون صفة (الافتراضي)	قيمته من النوع المتغير أى يمكن تغييرها بصورةٍ عادية، و حينها يكون بالإمكان أن نحجز له مكان في الذاكرة في أى وقت

مثال1:

```
حساب عام  الحساب.الأول = حساب()
```

و هنا نلاحظ أنه لو كتبنا:

```
حساب عام  الحساب.الأول = حساب()
الحساب.الأول = لا شئ
```

فإننا نكون بذلك قد أنهينا وجود الكائن فى الذاكرة و أصبح المتغير (الحساب.الأول) لا يشير إلى أى كائن.

الصيغة2:

```
اسم.الصنف مستوى.الوصول الاستنساخ  ثبات.المرجع  اسم.النسخة
اسم.النسخة = اسم.الصنف(المعاملات.إن.وجدت)
```

عدم استخدام = *اسم.الصنف(المعاملات.إن.وجدت)* يعنى أننا أشهرنا متغيراً من نوع الصنف و لكنا لم نحجز له مكاناً فى الذاكرة، ثم قمنا في السطر التالى بحجز ذلك المكان له.

مثال:

```
حساب عام  الحساب.الأول
الحساب.الأول = حساب()
```

تعبير في:

يُستخدم تعبير (في) حينما نستخدم متغير/اتٍ و/أو نستدعى إجراء/اتٍ فى كائنٍ ما لتوفير الوقت اللازم لكتابة الاستدعاءات منسوبةً إلي اسم كائن الصنف في كل مرة، و لإبقاء الكود مُختصَراً قدر الإمكان مع المحافظة على الوضوح، و صيغته كما يلى:

الصيغة1:

```
فى اسم.كائن:
```

```
\ إسنا د قيم لعنا صر الكائن /
متغير1=قيمة
متغير2=قيمة
```

مثال:

```
فى حساب1:
القيمة = 20000
الربح = 10
القيمة.الكلية = احسب()
```

الصيغة2:

```
فى كائن1  كائن2  ....:
\ إسنا د قيم لعنا صر الكائنات /
متغير1=قيمة
متغير2=قيمة
```

مثال:

```
فى حساب1  حساب2:
القيمة = 20000
الربح = 10
القيمة.الكلية = احسب()
```

ا ستخدام مكونات الكائن من صنف:

نستخدم العلامة _ لكى نستخدم أى مُكوِّنٍ من مُكوِّنات الأ صناف مثل المتغيرات و الإجراءات.

الصياغة:

```
الصنف_ا سم.المكون
```

مثال1:

```
الحساب.الأول_قيمة.الحساب = 50000
```

الوراثة فى الأ صناف:

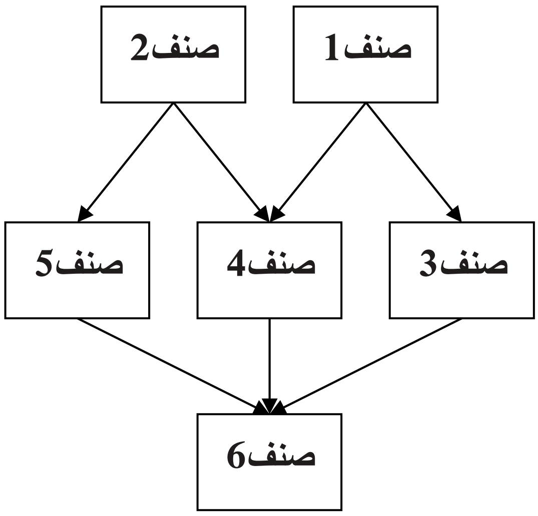
1- تدعم (الوراثة المتعددة multiple inheritance) و تُحَل مسألة تداخل أسماء المتغيرات و/أو الكائنات و/أو الثوابت و/أو الإجراءات و/أو الأصناف الداخلية و/أو الألقاب الداخلية (إذا حدث و تشابهت أسماء مُكوِّنٍ أو أكثر في صنفين موروثين و/أو صنفٍ موروثٍ مع الصنف الوارث) عن طريق:

1- جملة إعادة تسمية للأسماء المتشابهة و هي كما يلي:

صنف صنف.1 يرث صنف 2 صنف3:
تلقيب:
صنف3_اسم.المتغير.القديم = الاسم.الجديد1
صنف2_اسم.الإجراء.القديم(أنواع.المعاملات.لو.هناك تحميل) = الاسم.الجديد2

لو كان ذلك المُكوِّن إجراءاً فيجب كتابة أنواع المُعامِلات لو كان له تحميل. مع العلم بأن الإجراءات الموروثة في كل صنفٍ من الأصناف التي تغير اسم مكوناتٍ فيها سيتم تلقائياً تغيير كودها لتستخدم الأسماء الجديدة بدلاً من الأسماء القديمة.

أما مسألة كون أكثر من صنفٍ موروثٍ لها أصنافٌ مشتركةٌ يَرِثون منها جميعاً مثل الشكل التالي:



حيث أن الصنف (صنف6) يرث كلاًّ من الأصناف (صنف3) و (صنف4) و (صنف5). لكن المشكلة أن الصنفين (صنف3) و (صنف4) يشتركان في الوراثة من (صنف1) و أن الصنفين (صنف4) و (صنف5) يشتركان في الوراثة من (صنف2).

فلها أكثر من احتمالٍ و أكثر من حل:

1. **الاحتمال الأول:**

الأصناف الموروثة (صنف3 و صنف4 و صنف5) لم تُعدّل فى الأصناف الموروثة المشتركة بينها (صنف1 و صنف2)، و ليست بها أية إجراءاتٍ جديدةٍ زائدةٍ (أضيفت إليها زيادةً على الوراثة) متشابهة الأسماء، و هنا لا تكون هناك أى مشكلةٍ لأنه سيتم وراثة المتغيرات و الإجراءات و الأصناف التى تُوجَد فى الأصناف المشتركة (صنف1 و صنف2) كما هى فى الصنف الجديد الخاص بنا (صنف6)، و سيتم كذلك وراثة الإجراءات الزائدة فى (صنف3 و صنف4 و صنف5) كما هى بدون أى مشاكلٍ لأنها مختلفة الأسماء.

2. **الاحتمال الثانى:**

هناك تعديلٌ قد تم فى بعض أو كل الإجراءات الخاصة بالأصناف المشتركة، عن طريق قيام أحد الأصناف الوارثة لها بتغطية مُكوّنٍ من هذه المُكوّنات أو أكثر (أى التغيير فى بنائها كما سيتضح فى كلامنا عن الإجراءات)، أو هناك مُكوّناتٌ متشابهة الأنواع و الأسماء فى الأصناف الموروثة (صنف3 و صنف4 و صنف5)، و بالتالى تظهر مشكلة أن هناك مُكوّنات تتفق فى الاسم و لكنها تختلف فى البناء و من ثم لا نستطيع الاستغناء عن أحدها بالآخر.

و لهذا الأمر حلوله فى إبداع و هى كما يلى:

1) يدوياً باستخدام **تعبير الاستثناء من الوراثة:**

حيث يقوم المبرمج باستثناء كل المُكوّنات المتشابهة التى تُسبب المشكلة ما عدا مُكوّنٍ واحدٍ سيبقي لتتم وراثته هو فقط فى الصنف الجديد و من ثم تتلاشى المشكلة من الأصل.

و بالنظر إلى المثال السابق يمكننا أن نرى أن الحل سيكون كما يلى:

```
صنف صنف6 يرث صنف3 صنف4:
    ما عدا:
        صنف3_إجراء1
```

و لن يكون هناك أى تضاربٍ فى أسماء الإجراءات التى تمت وراثتها، و هكذا نرى أن الإجراءات التى ستُضَم نتيجة الوراثة إلى (صنف6) هى:

```
إجراء1() ----> من صنف4
إجراء2() ----> من صنف1
إجراء3() ----> من صنف1
```

و يجب كتابة توقيع الإجراء إذا ما كان هناك تحميلٌ له.

مثال:

```
صنف صنف6 يرث صنف3 صنف4:
```

```
ما عدا:
    صنف3_إجراء1(رقم نص)
```

2) يدوياً باستخدام التلقيب:

حيث يقوم المبرمج بنفسه بإعطاء كلِ المكونات المتشابهة أسماءً جديدةً ما عدا واحدٍ فقط. و عندئذٍ سيقوم المترجم تلقائيا بضمه إلى الصنف الجديد لأنه وحيد الاسم و ليس هناك مُكوّناتٌ أخرى تُشاركه ذلك الاسم بعد عملية التلقيب أى أن المشكلة لن تظل موجودة. و ستقوم المُكوّنات التى لم تُغطّى بندائه هو فى كودها لأنه الوحيد الذى له الاسم القديم.

مع الأخذ فى الحسبان أنه عند تلقيب جميع المُكوّنات المشتركة و تغيير أسمائها و عدم ترك أيٍ منها باسمها القديم، فإن المُكوّنات الأخرى التى لم تُغَطَّى فى الأصناف الموروثة ستظل فى هذه الحالة أيضاً تُنادِى المكونات المشتركة بأسمائها القديمة، و من ثم سيُخرِج لنا المترجم رسالة خطأٍ و لن يُكمِل عمله، و ذلك لأنه لن يكون هناك مُكوّنٌ بنفس الاسم القديم الذى تستخدمه المكونات غير المُغَطّاة.

و بالتطبيق على مثالنا نرى أن الحل سيكون كما يلى:

```
صنف صنف6 يرث صنف3 صنف4:
    تلقيب:
        صنف3_إجراء1 = إجراء3.1
```

فتكون الإجراءات التى تُضَم إلى صنف6 كما يلى:

```
إجراء1()   ----> من صنف4
إجراء2()   ----> من صنف1
إجراء3()   ----> من صنف1
إجراء3.1() ----> من صنف3
```

و عندما يقوم الإجراءان (إجراء2 و إجراء3) باستخدام (إجراء1) سيجدان أمامهما الإجراء المضموم من (صنف4) فقط و ليس شيئاً آخر.

ملحظٌ هام: فى هذه الحالة فإن الإجراءات التى سيتم تلقيبها سوف تستدعى نفسها (لو كانت تستدعي نفسها) بالإسم الجديد.

3- إذا ما تم تعريف إجراءٍ فى الصنف الوارث له بنفس ا سم إجراءٍ آخرٍ فى الصنف الموروث فإن الإجراء الجديد سيُغَطِّى القديم و يُخفيه عن المستخدم للصنف الوارث، و يظهر له الإجراء الجديد فقط. و لا يُحتَاج إلى كتابة كلماتٍ خا صةٍ تُعبر عن قابلية الإجراء فى الصنف لتغطيته فى الأ صناف التى ترث صنفه، بل هناك كلمةٌ مفتاحيةٌ واحدةٌ لجعل الإجراء غير قابلٍ للتغطية و هى كلمة (سقف) كما سيلي في الجزء الخاص بالإجراءات.

4- لا يُسمح بوراثة أ صنافٍ لها أ سماءٌ متشابهة.

الكلمة المفتاحية (هذا):

هى كلمةٌ مفتاحيةٌ تُستخدَم للإشارة إلى الصنف الذى تُكتب داخله، و نلجأ إليها عندما نُريد ا ستخدام مُكوّنٍ من مُكوّنات الصنف الحالى و هناك مكونٌ آخرٌ يحمل نفس الاسم فى نفس المساحة التكويدية code block، أى أن هناك مشكلة تضاربٍ فى الأسماء ستحدث.

و مثالٌ على ذلك أن يكون هناك فى المُعامِلات المُمرَّرة لإجراءٍ متغيراتٌ لها أ سماءٌ تُشبه أ سماء المتغيرات التى تُوجد فى الصنف الحالى، و بالتالى عند ا ستخدام ا سم المتغير يتم ا ستخدام المُعامِل لا متغير الصنف رغم أنه ربما يكون هو المقصود بالاستخدام، و تُحل هذه المشكلة با ستخدام الكلمة هذا كما فى المثال التالى:

```
صنف صنف.1:
    نص المحتوى
    إجراء غير.النص.إلي( نص المحتوى):
        هذا_المحتوى = المحتوى
```

المُشيِّدات:

تُماثِل الـconstructors في اللغات الأخري، و هى إجراءاتٌ عاديةٌ يتم ا ستدعاؤها عندما يتم تعريف كائنٍ جديدٍ من الصنف المُحتوِى على المُشيِّد و حجز مكانٍ له فى الذاكرة، و فى حالة لغة إبداع يتم هذا عندما يُكتَب:

= ا سم.الصنف ()

بعد ا سم الكائن فى جملة تعريف الكائن الجديد، و قد سبق شرح هذا.

و المُشيِّدات فى إبداع هي إجراءاتٌ عاديةٌ تماماً و لكن يُشترط فيها شرطٌ واحدٌ فقط هو أن يكون ا سمها هو نفس ا سم الصنف الحاوي لها.

و للمُشيِّدات نوعان:

1- ***الافتراضي***: أى الذى لا نُمَرِّر له أى مُعامِلات.

مثال:

```
إجراء عام  صنف1:
هذا_الحساب = 2000
```

-2

ذو المعاملات:

مثال:

```
إجراء عام  صنف1(رقم الحساب):
هذا_الحساب = 2000
```

و ا ستدعاء مُشيّدات الأ صناف الموروثة عند ا ستدعاء مُشيّدات الأ صناف الوارثة يكون طبقاً للقواعد التالية:

- في حالة وجو د مُشيّداتٍ تُماثِل مُشيّد الصنف الوار ث في توقيع المُدخَلات:

يقوم المُترجم تلقائيا با ستدعائها في كـل مـرةٍ يُسـتدعي فيهـا مُشـيّد الصـنف الـوارث، و تكـون قيـم المُعامِلات المُمَرّرة لكل مُشيّدٍ منها هي قيم المُعامِلات المُمَرّرة لمُشيّد الصنف الوارث.

- في حالة عدم وجـو د و لـو مُشـيّدٍ واحـدٍ فـي صـنفٍ مـن الأ صـناف الموروثـة يحمـل نفـس توقيـع المدخلات الذي لمُشيّد الصنف الوارث:

فإن المترجم يترك أمر ا ستدعاء مُشيّدات الأ صناف الموروثة للمبرمـج تمامـاً لتكـون العمليـة يدويـةً بالكامل.

خُيُوط التنفيذ executing threads

(لم يتم بناؤها في الإ صدارة 1.0 من مُفسِّر اللغة).

في إبداع تُعتبَر خيوط التنفيذ المتعددة عبارة عن إجراءاتٍ يتم تنفيذ كلٍ منها بالتوازي مع الأُخرَيات، و تُعامَل كحالةٍ وَسيطةٍ بين كائن الصِنْف و الإجراء العادي، كما سيتضح من الشرح التالي.

إنشاء الخيط:

أنشيء("اسم.الخيط.الجديد" اسم.الإجراء(التوقيع إن احتجناه))

مثال:

أنشيء("الخيط.الأول" إجراء.ما)

الإستخدام:

لبدء عمل الخيط يتم التعامل معه كأنه إجراء عادي تماماً، كما يلي:

اسم.الخيط(المُعامِلات)

مثال:

أنشيء("أوجد.الرقم.الأكبر" الرقم.الاكبر) رقم أكبر.رقم = أوجد.الرقم.الأكبر(542 9898)

التحكم في عمل الخيط:

يُمكِن التحكم في عمل الخيط عن طريق تطبيق إجراءٍ قياسيٍ عليه (بما يُشبِه التعامل مع كائنات الأصناف)، كما يلي:

إسم.الخيط_الإجراء.القياسي(المُعامِلات)

مثال:

الخيط.الأول_أوقف(1 2)

الإجراءات القياسية التي يُمكِن تطبيقها على الإجراء الخيط:

ملحظٌ هام: أسماء الإجراءات القياسية التالية مُجرَّد خطوطٍ عريضةٍ لما سيُوجَد عند بناء خيوط التنفيذ في مُفسِّر اللغة، أي أنها قد تتغير و/أو تزيد و/أو تنقص.

- أوقف()

  لإيقاف تنفيذ خيط التنفيذ الحالي نهائياً

- أوقف.الكل()

لإيقاف تنفيذ كل خيوط التنفيذ الحالية نهائياً

- أثبط(فترةزمنية)

لإيقاف تنفيذ خيط التنفيذ لفترة معينة

- أنشط()

لإعادة خيط تنفيذ إلى العمل بعد التثبيط

- أثبط.الكل(فترةزمنية)

لإيقاف تنفيذ كل خيوط التنفيذ ما عدا صاحب النداء لفترة معينة

- أنشط.الكل()

لإعادة كل خيوط التنفيذ المثبطة إلى العمل

الجملة الشرطية (لو)

الصيغة:

```
لو \ الشرط /:
    \ الأوامر /

أما لو \ شرط آخر/:
    \ الأوامر /

غيره:
    \ الأوامر /
```

و الأقواس اختيارية

مثال:

```
لو س = ص :
    أكتب.نص. سطر("س يساوي ص")

أما لو س في من 1 إلي 12 :
    أكتب.نص. سطر("س في المجموعة من 1 إلي 12")

أما لو س في {1 2 3 4 5} :
    أكتب.نص. سطر("س في المجموعة {1 2 3 4 5}")

أما لو س في جدول.الأرقام :
    أكتب.نص. سطر("س في جدول.الأرقام")

أما لو س > 12 و س < 20 :
    أكتب.نص. سطر("س بين الـ12 و الـ20")

غيره:
    أكتب.نص. سطر("س خارج الاحتمالات الممنوحة")
```

حيث نري أن التعبيرات المنطقية المسموح بها كما يلي:

- **التعبيرات العا دية** مثل:
    - س = ص

- ○ س > 12 و س < 20

- **تعبير في:** و هو دائماً علي الشكل التالي:

<حاوية> في <تجميعٌ من الحاويات من نفس النوع>

و له الحالات التالية:

- ○ **في جدول:** أن يكون التجميع عبارة عن إسم جدولٍ أو بُعدٍ في جدول، مثل:

لو س في جدول.الأرقام

- ○ **تعبير** (من – إلي – بقيمة –) : و لا يصح استخدامه إلا مع القيم الرقمية، مثل:

لو س في من 2 إلي 100 بقيمة 2

- ○ **قيم مبا شرة**، مثل:

لو س في {1 2 3 4 5 6}

- **المزج بين كل ما سبق**، مثل:

لو س > 12 و س < 20 أو س في {1 2 3 4 5}

الجداول

الصياغة1:

التعريف الكامل بدون تحديد المحتويات:

النوع{عدد الأبعاد} مستوى-الوصول الإستنساخ ثبات.المرجع الاسم1 الاسم2 ....

الصياغة2:

التعريف مع تحديد المحتويات:

```
النوع{عددالأبعاد} مستوى-الوصول الإستنساخ الاسم = {                        |
اسم.البعد.الأول = {     اسم.الصف.الأول = \عناصر الصف/                      |
                        اسم.الصف.الثانى = \عناصر الصف/                     |
                        اسم.الصف.الثالث = \عناصر الصف/        }            |
اسم.البعد.الثاني = {    اسم.الصف.الأول = \عناصر الصف/                      |
                        اسم.الصف.الثانى = \عناصر الصف/                     |
                        اسم.الصف.الثالث = \عناصر الصف/        }            |
\بقية الأبعاد/
}
```

مع ملاحظة أنه فى كل الصِيَغ الفائتة فإن:

1. أسماء الصفوف يمكن إهمالها مع علامة التساوى التى تليها.
2. لو أعطينا الجدول النوع (حر) فإن كل صفٍ يُمكِن أن يكون له نوعٌ خاصٌ به، أما لو كان للجدول نوعٌ آخرٌ غير (حر) فلا يمكن لأى صفٍ أن يحوى غير ذلك النوع، و لا يمكن كتابة نوعٍ أمام اسم الصف.

مستويَات الوصول:

إذا استُخدِمت مع الجداول التى توجد فى الأصناف يكون لها نفس معاني مستويَات وصول المتغيرات التى توجد فى الأصناف.

الاستنساخ:

إذا استُخدِمت مع الجداول التى توجد فى الأصناف يكون لها نفس معاني صفات استنساخ المتغيرات التى توجد فى الأصناف.

ثبات المرجع:

تماماً ككائنات الأ صناف. و يجب أن يكون الجدول ثابتاً لكي يمكن إعطاء أبعـا ده و صـفوفه أ سـماءاً خا صـةً بها.

مثال1:

```
رقم{1} عام مشترك الأعدا د
```

مثال2:

```
رقم{2}عام ثابت الأعدا د = { الصف.الأول={4 6 2} الصف.الثانى={33 16 10}}
```

مثال3:

```
رقم{2} عام الأعدا د
الأعدا د = {{33 16 10} {4 6 2}}
```

الجداول ذات الأبعا د مختلفة الطول:

للمبرمج إمكانية إ شهار جداولٍ ذات أبعا دٍ مختلفة الأطوال.

مثال:

```
رقم{2} عام مشترك الأعدا د = {{14 8 16} {5 46 8 4 6 2}}
الأعدا د{4 1} = 20
أكتب.رقم. سطر(الأعدا د{4 1})
```

حجز أماكنٍ فارغةٍ لعنا صر الجدول:

إذا أر دنا أن نحجز أماكناً فارغةً لعنا صر الجدول يمكننا أن نفعل ذلك كما يلي:

```
ا سم.الجدول = جديد(عد د.عنا صر.البعد.الأول عد د.عنا صر.البعد.الثاني ....)
```

مثال:

```
رقم{2} الأرقام = جديد(10 20)
```

و سيتم إعطاء تلك العنا صر قيمها الافتراضية حسب نوعها بمجر د حجز مكانٍ لها.

الإ ستخدام:

و للو صول إلى عنصرٍ في جدولٍ فهناك طريقتان لفعل ذلك هما:

1- بالمُمَيِّزات indices:

نكتب الأرقام التى تعبر عن ترتيب العنصر فى الجدول بين أقواسٍ من النوع { }، و لا يُفصَل بين رقم إحداثى كل بُعْدٍ و الآخر بأي فا صلٍ، كما يلى:

اسم.الجدول{إحداثى.البعدالأول إحداثى.البعدالثانى ......}

مع الأخذ فى الحسبان أن العد يبدأ فى الجداول فى لغة إبداع بالمميز 1.

مثال:

رقم{2} الأعداد = { { 2 6 4 } { 5 8 7 } } الأعداد{ 1 3 } = 15

و هنا قمنا بتغيير قيمة العنصر ذى الصف الأول و العمود الثالث و الذى كان يحمل القيمة (4) إلى القيمة (15).

و للمبرمج إمكانية تحديد جزءٍ معينٍ للعمل عليه من الجدول با ستخدام تعبير (من – إلي -)، كالتالى:

مثال:

الأعداد{من 1 إلي 10 1} = الأعدادالأخرى{5 من 10 إلي 20 }

و في هذا المثال وضعنا الأعداد الموجودة في الصف الخامس من الجدول المُسمَّي (الأعدادالأخرى) بدايةً من العنصر رقم 10 إلى العنصر رقم 20 في العمود الأول من الجدول المُسمَّي (الأعداد) بداية من الصف رقم 1 إلى الصف رقم 10.

2- طريقةٌ أخرى للاستخدام:

يمكننا التعامل مع الصف كأنه جدولٌ كاملٌ (ذو بُعْدٍ واحدٍ)، كما يلي:

اسم.الجدول_اسم.الصف{ }

مع ملاحظة أنه لديه ميزةٌ أخرى، هي أنه لو أردنا الحصول على قيمةٍ في الصف المسمى (أ) و التي تُناظِر القيمة (ق) في الصف (ب) فإنه يمكننا القيام بالآتي:

الجدول _ أ (ب = ق)

و هو ما يعني (قيمة الصف **أ** المُوازِية للعنصر ذي القيمة **ق** في البُعد **ب**).

معرفة طول الجدول أو بُعدٍ من أبعاده:

يُمكن ذلك عن طريق استدعاء الإجراء طول() كما يلي:

اسم.الجدول_طول()

معرفة عدد الأبعاد لجدولٍ أو لبُعدٍ من أبعاده:

يُمكن ذلك عن طريق استدعاء الإجراء عدد.الأبعاد() كما يلي:

اسم.الجدول_عدد.الأبعاد()

المُتَسَلسِلَات العددية:

التعبير

{ من قيمة1 إلى قيمة2}

يعيد جدولًا أُحادي يبدأ بالعنصر (قيمة1) و ينتهي بالعنصر (قيمة2) و يزيد كل عنصر عن ما يسبقه بمقدار (1). و لو أردنا تغيير مقدار الفرق بين كل عنصرٍ و الذي يليه نستخدم الصياغة التي في المثال التالي:

{من قيمة1 إلى قيمة2 بقيمة قيمة3}

و هذا التعبير يعيد جدولًا أحاديًا يبدأ بالعنصر (قيمة1) و ينتهي بالقيمة (قيمة2) و كل قيمة جديدة تزيد عن القيمة القديمة بمقدار (3).

و يمكن استخدام هذا التعبير لملء الجداول كالتالي:

مثال1:

رقم{2} الأعداد = {{من 4 إلى 6 } {5 8 7}}

مثال2:

رقم{2} الأعداد = { { من 4 إلى 8 بقيمة 2 } { 5 8 7 } }

الجُمَل التِكرارية

جملة بينما الشرطية:

```
تكرار الاسم:
بينما \ شرط / :
\ الكود /
```

مثال:

```
تكرار التكرار.الأول:
بينما س < 15 :
أكتب.نص.سطر(س)
س = س+1
```

جملة بينما للتتابعات:

```
تكرار \ الاسم / :
بينما \ اسم.الحاوية / في \ اسم.التجميعة / :
\ الكود /
```

و التجميعات هنا هي نفسها التي ذكرناها في شرح جملة لو الشرطية.

مثال1:

```
تكرار التكرار.الأول::
بينما س في الأعداد:
أكتب.نص.سطر(س)
```

مثال2:

```
تكرار التكرار.الثاني:
بينما س في {من 10 إلى 20}:
أكتب.نص.سطر(س)
```

إجراءاتٌ قياسيةٌ ذات صلة:

1- أتخطى(اسم.التكرار):

و تُستَخدم فى الحالات التى نُريد من المترجم أن يُهمِل تنفيذ الأوامر التى تليها فى صُلب الجملة التكرارية و يقوم بتنفيذ الدورة التكرارية التالية. و تُكافِئ كلمة continue فى عائلة الـC.

مثال: للبرنامج التالى:

```
رقم{1} ص = { 1 2 3 4 5 }
التكرار التكرار.الأول:
بينما س فى ص :
    لو س = 3 :
        أتخطى( التكرار.الأول)
    أكتب.نص. سطر(س)
```

سيكون الخرج كما يلى:

```
1
2
4
5
```

و إذا لم نكتب اسم حلقةٍ تكراريةٍ داخل قوسي الأمر أتخطي فسيَفترض المترجم أنه اسم آخر حلقةٍ تكراريةٍ من اللواتي كُتِب داخلهن.

مثال: للبرنامج التالى:

```
رقم{1} ص = { 1 2 3 4 5 }
التكرار التكرار.الأول:
بينما س فى ص :
    لو س = 3 :
        أتخطى( )
    أكتب.رقم. سطر(س)
```

سيكون الخرج كما يلى:

```
1
2
4
5
```

2- أنهي(اسم.التكرار):

و تُكافِئ كلمة break في عائلة الـC، حيث تُستَخدم لكسر تنفيذ الجملة التكرارية، و تُعامَل كأمر أتخطي تماماً.

و مثالٌ على ذلك البرنامج التالي:

```
رقم{1} ص= { 1 2 4 0 3 5 }
التكرار التكرار.الأول:
بينما س في ص :
    لو س = 0 :
        أنهي(التكرار.الأول)
    أكتب.نص.سطر( 1÷ س )
```

سيكون الخرج كما يلي:

```
1
0.5
0.25
```

و هناك أمورٌ هامةٌ بخصوص استعمال هذين الإجرائين:

- لا يمكن استعمالهما في المسار العام للبرنامج و لا في داخل الإجراءات مباشرةً، بل يمكن استعمالهما فقط داخل تعبيري لو و بينما.
- عند كتابة أي من هذين الإجرائين في كتلة أوامرٍ معينة فإنه لا يمكن ان يأتي بعدهما أي أمرٍ آخرٍ في ذات كتلة الأوامر.

مثال:

```
بينما س في ص :
    لو س = 0 :
        أنهي(التكرار.الأول)
        س = 5                هذا خطأ
```

الألقاب

نبذةٌ بسيطة:

هى مُكوّناتٌ تُمَكِّننا من ضم مجموعةٍ من الكلمات على أساس أنها تُمَثِّل قيماً ضمن نوعٍ مُعينٍ من أنواع البيانات هو اللقب، فكما نقول أن النوع رقم يضم القيم

5 4 3 2 1

و يُمكِننا إشهار متغيرٍ من النوع رقم له القيمة 3 فإنه يُمكِننا باستخدام الألقاب القول بأن النوع (يوم) يضم القيم (سبت أحد إثنين) حيث يمكن إشهار متغيرٍ من النوع (يوم) له القيمة (سبت) على سبيل المثال، و استخدام ذلك فى الكود. و بالتالي هي تُماثِل الـEnumeration فى عائلة الـC و التى يُترجمها كل من قرأت لهم بالكلمة (تعدادات) و التى لم أستسغها و فَضّلتُ عليها كلمة الألقاب الواضحة المعنى و الأصغر حجما.

الصياغة:

يمكن إشهار لقبٍ جديدٍ باستخدام الصِيَغ التالية:

الصيغة1:

لقب مستوى.الوصول الاسم : العنصرالأول ..... العنصررقم.ن

مستوَيات الوصول:

حينما تُستخدَم مع الألقاب التي في الباقات أو في الأصناف يكون لها المعاني التالية:

المستوى	الشرح
عام (الافتراضي)	يمكن الوصول إليه من أى مكان
داخلي	إن كان في صنفٍ فيُمكن الوصول إليه من داخل صنفه و الأصناف الوارثة له فقط. أما إن كان في ملف باقةٍ فيمكن الوصول له من داخل باقته فقط.
خاص	إن كان داخل صنفٍ فيمكن الوصول إليه من داخل صنفه فقط. أما إن كان داخل باقةٍ فيمكن الوصول له من باقته فقط.

مثال:

لقب خاص يوم:
السبت الأحد الإثنين الثلاثاء الأربعاء الخميس الجمعة

صِيَغ الاستخدام:

صياغة:

اسم.اللقب مستوى.الوصول الإستنساخ الثبات اسم.الكائن = اسم.اللقب_اسم.العنصر

مستويات الوصول:

إذا استُخدِمت مع كائنات الألقاب التي تعرف فى الأصناف: فكائن اللقب له نفس مستويات وصول متغيرات الأصناف و نفس معانيها.

الاستنساخ:

إذا استُخدِمت مع كائنات الألقاب التي تعرف فى الأصناف: فكائن اللقب له نفس صفات متغيرات الأصناف و نفس معانيها.

الثبات:

تماماً كموا صفات المتغيرات.

مثال:

يوم عام مشترك يوم أحدالأيام = يوم_الخميس يوم.الإجازة = يوم_الجمعة

تحويل قيمة كائن اللقب إلي نص:

قد نحتاج لهذا لو أردنا أن نكتب قيمة كائن اللقب علي الشاشة مثلاً، و يُمكننا هذا عن طريق كتابة:

اسم.اللقب_إلي.نص(القيمة)

أمثلة :

يوم_إلي.نص(أحدالأيام)
يوم_إلي.نص(يوم_الجمعة)

مُعالَجة الأغلاط

نبذةٌ بسيطة:

فى إبداع يُطلَق الاسم غلط على ما يُسميه الآخرون (استثناء) و كلاهما ترجمةٌ لكلمة exception. و قد استَخدمتُ كلمة غلط لأن الاستخدام الغالب على هذا النوع من الأدوات يكون لمعالجة ما يعتبره المبرمج خطأً منطقياً أثناء زمن التشغيل و هي الأمور التى يَصِح لغوياً إطلاق و صف الأغلاط عليها؛ لأن لفظة غلط تُطلَق على الشئ الذى يُوضَع فى غير محله. و بذلك تُعبّر عن المعنى جيداً و فى نفس الوقت هى أخف و أصغر من لفظة استثناء التى تزيد حروفها عن حروف غلط و تتناثر أكثر على لوحة المفاتيح.

تعريف غلطٍ جديد:

حينما نُريد أن نُعرّف أمراً ما على أساس أنه غلطةٌ منطقيةٌ: يُمكننا أن نفعل ذلك بالخطوات التالية:

1) تعريف صنفٍ جديدٍ يرث من الصنف (غلط).

مثال:

```
صنف غلط1 يرث غلط:
    نص السبب

    إجراء إلى.نص:
        أكتب.نص.. سطر("هذا غلط من النوع غلط1")

    إجراء تنبيه:
        أكتب.نص. سطر("هذا تنبيه على وقوع الغلط فى البرنامج بسبب أن " + السبب)
```

2) وضع جملة إعلان وجود غلطٍ فى الأوامر المُراد معالجة الغلط فيها، و نستخدم الكلمة المفتاحية (أعالج) يليها اسم نسخةٍ من الغلط المراد معالجته.

و يمكننا ضبط متغيرات الكائن قبل أمر المعالجة بحيث يستفيد منها المُعالِج عند القيام بالمُعالَجة كما فى المثال التالى.

مثال:

```
لو س > ص :
    غلط1 غ = غلط1()
```

```
غ_السبب = "المتغير س > المتغير ص"
أعالج(غ)
```

3) نُحيط جملة الإعلان عن الغلط بتعبير معالجة الأغلاط، و هو تعبير (أحاول) التالي شرحه.

مثال:

```
أحاول:
    لو س > ص :
        غلط1 الغلط = غلط1()
        الغلط_السبب = "المتغير س > المتغير ص"
        أعالج(الغلط)
    ع = ص - س

معالجة غلط1 غ:
    غ_تنبيه()
```

صياغة معالجة الأغلاط:

```
أحاول :
    \ الأوامر المرا د تنفيذها عا دة /
معالجة نوع.الغلط الاسم:
    \ الأوامر التى ستنفذ فى حالة حدوث الغلط المكتوب فى الأعلى /
معالجة نوع.الغلط الاسم:
    \ الأوامر التى ستنفذ فى حالة حدوث الغلط المكتوب فى الأعلى /
.

دوماً:
    \ الأوامر التى ستنفذ لو حدث غلط أم لم يحدث /
```

مثال:

```
أحاول :
    أكتب.نص. سطر("تجربة")
```

```
معالجة غلط.دخل.خرج س :
        س_إلى.نص()
```

في حالة رفع غلطٍ ما و وجود أكثر من فرعٍ من أفرع (معالجة) فسيتم اختيار الفرع الأكثر تناسباً مع نوع الصنف الخاص بالغلط تلقائياً، تماماً كما يحدث عند نداء إجراءٍ مُحَمَّل.

مثال: البرنامج التالي:

```
أحاول:
        تجربة.ر د.الفعل()

معالجة غلط  غ :
        أكتب.نص. سطر("معالجة غلط  غ")

معالجة غلط1 غ :
        أكتب.نص. سطر("معالجة غلط1 غ")

دوما:
        أكتب.نص. سطر("تنفيذ دوماً")

إجراء تجربة.ر د.الفعل:
        غلط1 غلطة = غلط1()
        غلطة_الرسالة = "الغلط  ادا حصل بمزاجي"
        أعالج(غلطة  صحيح)

صنف  غلط1 يرث  غلط  :
        إجراء غلط1 :
            لا شيء
```

له الخرج التالي:

```
معالجة  غلط1 غ
```

تنفيذ دوما

الأغلاط واجبة المُعالجة:

في بعض الأحيان قد نحتاج إلي جعل معالجة الأغلاط التي قد تحدث عند تنفيذ إجراءٍ ما أمراً لازماً للمبرمج الذي سوف يستدعي ذلك الإجراء، و لذلك يمكننا تمرير القيمة المنطقية صحيح لأمر المُعالُجة أعالج حتي تؤدي هذا الدور.

و مثالٌ علي ذلك البرنامج السابق الذي كُتِب فيه:

أعالج(غلطة صحيح)

لكي يفرض علي من يستدعي الإجراء (تجربة.ر د.الفعل) أن يحيط أمر الاستدعاء بتعبير أحاول فإن لم يفعل فإنه يحصل علي خطأٍ نحويٍ من المترجم.

الأغلاط القياسية في إبداع:

في كل لغة برمجةٍ تدعم معالجة الأغلاط يكون فيها مجموعةٌ من الاغلاط القياسية التي من الممكن للمُبرمِج الإعتماد عليها في أكواده، مثل (غلط القسمة علي الصفر) أو (غلط محاولة الوصول إلي عنصرٍ في المصفوفة باستخدام مميزٍ خارج نطاق حدودها). أما في الإصدارة الأولي من إبداع فلم يتم بعد تصميم تلك الأغلاط القياسية أو بناؤها. علي أنه من المُخطَّط له أن يتم تصميمها الواحدة تلو الأخري و بناؤها بإذن الله تعالي في الإصدارات القادمة.

تنبيه هام: يجب التنبه إلي أنه لا يمكن كتابة أي كتابة أي أكوادٍ بعد أمر أعالج في نفس كتلته التكويدية code block و إلا اعتُبِر هذا خطأ.

مثال:

لو صحيح:

أعالج (غلط())

أكتب.نص. سطر("هذا خطأ") ✗

إ ستخدام الأكوا د الخارجية في إبداع

الأكوا د الخارجية في إبداع تنقسم إلى الفئات التالية:

- ملفات الباقات العا دية التي تحتوى على أكوا د أ صناف:

و يُمكِن ا ستخدامها كما قلنا من قبل عـن طريـق أمـر أضـم ثـم التعامـل مـع الباقـات الداخليـة و/أو الأ صناف و/أو الألقاب التي بداخلها بصورةٍ عا ديةٍ جداً.

- ملفات مكتبات الربط الديناميكي (مثل الـDLL في الويندوز و الـSO في القنو/لينـوكس) و الـتي هي عبارة عن أكوا دٍ مكتوبةٍ بإبداع ثم تم ترجمتها لكى تكون مكتبة مُتنقلةً قابلةً للا ستخدام فـي زمـن التشغيل:

و هذه تُعامَل كما تُعامَل ملفات الباقات العا ديـة تمامـاً، أى نسـتخدم معهـا الأمـر أضـم، ثـم تُسـتَخدَم مُكوِّناتها الداخلية كالعا دة.

(لم يتم بناؤها في الإ صدارة 1.0 من مُفسِّر اللغة).

- واجهات أنظمة التشغيل المُختلفة API التي ربما يو د المُبرمِج الو صول إليها في برامجـه: و هـذه لم يتم الإ ستقرار علي الموقف منها في إبداع (في هذه الإ صدارة من المُفسِّر علي الأقل).

رُوح الإبداع

فى هذا الجزء سوف أُناقش أهم القضايا البرمجية التى ركَّزتُ عليها بشدةٍ أثناء تصميم إبداع. و ربما يَلمس القارئ بنفسه هذه القضايا و يَستنتِجها جميعها لوضوح الإهتمام بها فى الشروحات السابقة للّغة، و سر تركيزى على هذه القضايا أكثر من غيرها يَعكِس اقتناعى الشديد بأنها ذات أهميةٍ قُصْوى فى العمل البرمجى عامةً و فى تصميم لغات البرمجة خاصة، و ذلك حيث أن لغة البرمجة هى الأداة التى بها يتم بناء باقى مُخرجَات العمل البرمجى فبالتالى هى المكون الأكثر عكساً للقضايا الهامة برمجياً.

و فى هذه القضايا قد أُخالِف الكثيرين من أبرز أهل الإختصاص فى آرائهم و قَناعاتهم، و ليس الأمر أننى من مُحبِّى التفرد و الغَرابة اللذين يفعلون ما يفعلون من أجل لفت الأنظار فقط، بل إننى فَعلتُ ما فَعلتُ عن قناعةٍ حقيقيةٍ وَلَّدها فِىّ تأملٌ كبيرٌ فى العلم البرمجى و ما أُحِب له أن يصير إليه، و كذا ما آخذه على الواقع الحالى من مآخِذٍ من الأفضل أن تُزال، و ما أراه فيه من حَسناتٍ حبّذا لو تُقَوّى و تُزاد.

و مَرَدُّ الأمر فى النهاية إلى القَناعات الذهنية الذاتية، و ليس هذا المجال من المجالات التى يُمكِننا فيها القول بالخطأ المَحْض و الصواب المَحْض، بل الأمر هنا هو الإجتهاد البشرى الذى يُصِيب صاحبه و يُخطِئ، و ليس لأحدٍ تحجيم آراء أحدٍ أو منعها أو تقييد الحرية فى التّفَوُّه بها و نَشْرِها ما دام الأمر أولاً و آخِراً جهداً بشرياً خالصاً قابلاً للنقد و الدحض.

و قد كان من المُمكِن أن أُفْرِد لهذا الجزء ورقةً علميةً خاصةً به أتوسع فيها فى تِبيان وجهات نظرى فى مسائلٍ تتعلق بتصميم لغات البرمجة، إلا أننى رأيتُ أنه لابد من إدراجه هنا بعد ذِكْر مُكوِّنات لغة إبداع و قبل ذِكر ما ليس من مُكوِّناتها؛ حتى يُبَيِّن و يُوضِّح الأساس الذى بَنيتُ عليه كل ما سبق و كل ما سيأتى.
ففى هذا الجزء سيتضح السبب وراء إدراج كل مُكوِّنٍ من مُكوِّنات اللغة، و سيتضح كذلك القَناعة التى دفعتنى إلى رفض ضم أى مُكوِّنٍ رَفضتُ ضمّه إلى اللغة، و بالتأكيد سيتضح للقارئ ما غَمُضّ عليه فى شرح أسباب رفض المُكوِّنات فى الجزء التالى بسرعةٍ كبيرةٍ لفهمه لنظرتى للغات البرمجة جيداً.

و القضايا التى سأُركِّز عليها هنا هى على الترتيب:

1) التخصُّص،
2) الوراثة،
3) التهجين،
4) الشمول،
5) البساطة و الاستقرار،
6) الأمن،
7) الأُلْفة.

التخصُّص:

قضية التخصُّص أحصرُها هنا فى المسألة التى تُناقِش ما يجب على المُبرمِج البانى للمُـترجِم القيـام بـه و مـا يجب أن يُترَك للمُبرمِج مُستخدِم اللغة لكى يفعله، و يُمكِنني أن أُعَبِّر عن رأيى فى هذه القضية بجُملةٍ بسيطةٍ و هى **" دع الإدارةَ للمُترجِم و حَلَّ المشاكل الأساسية للمُستخدِم".**

و حينما أقولُ الإدارة فإننى أقصد مُجمَل الأنشطة الـتى تختـص بإدارة الـذاكرة، و الـتى أُفَصِّـلها فـى البنـدين التاليين:

- تحديد المكان الذى ستُوضَع به البيانات فى الذاكرة، أى عملية حجز الأماكن لها.
- تحديد الوقت المُناسِب الـذى سـيتم فيـه إرجـاع الأمـاكن الـتى حُجِـزت لمُتغيِّراتٍ فـى الـذاكرة (ثـم لـم تَعُـد هنـاك حاجـةٌ لهـا) إلـى حَـوْزة نظـام التشـغيل مـرةً أخـرى لإعـادة استغلالها.

فهاتان النقطتان تُمَثِّلان التعامل الكامِل مع الذاكرة من وجهة النظر البرمجيـة للمُسـتخدِم لنظـام التشـغيل، أمـا الإدارة التى يقوم بها نظام التشغيل فهى عمليةٌ أكثر تعقيداً و تشـعباً مـن هـذه بكـثير، و إن كـانت ترتكِـز عليهـا بشكلٍ أساسى.

و نظرتى إلى هاتين النقطتين على أنهما من نصيب المُبرمِج البانى للمُترجِم أو المُفسِّر يُوضِّح نظرتى إلـى بنيـة إدارة الذاكرة و توزيع المسئوليات وفقها، حيث أُقسِّمها إلى ثلاثة أقسامٍ هى:

1) القِسْم الأدنى:

و يختص به نظام التشغيل، و هي الأمور الـتى تتعلـق بكيفيـة تنظيـم العمليـات الـتى يجـرى تنفيـذها علـى الحاسـوب فـى الـذاكرة، و كيفيـة إحْلال البرامـج الجديـدة الـتى يرغـب المُستخدِم للحاسوب فى العمل عليها محـل تلـك الـتى قيـد التنفيـذ و لكنهـا مُتوقِّفـةٌ عـن العمل وقتياً.

و كذا كيفية التنسيق بين التصور للذاكرة على أنهـا منطقـةٌ مُتصِـلةٌ مـن الأمـاكن التخزينيـة مصفوفةً فوق بعضها البعض، و بين التصور الـواقعى لهـا علـى أسـاس البِنـاء العَتَـادى لهـا (أى طريقة بِنائها كما صمَّمتها شركة التصنيع لرقاقة الذاكرة).

2) القِسْم الأوسط:

و يختص به المُترجِم و المُفسِّر، و يختص بحجز مكانٍ فـى الـذاكرة للمُتغيِّـرات و مُكوِّنـات البرنامج المُختلِفة، و فى النهاية إرجاع أى مِساحةٍ فى الذاكرة لم يعد البرنامـج فـى حاجـةٍ إليها إلى نظام التشغيل مرةً أخرى؛ لتقليص المفقـود مـن إمكانيـات جهـاز الحاسـب إلـى أدنى المراتب.

3) القِسْم الأعلى:

و هو مُجرَّد الإستخدام بدون التقييد بكيفيةٍ مُعيَّنة، و يختص به بطبيعة الحال المُبرمِج العادى الذي يَستخدِم لغة البرمجة.

و أنا أَعتبِر أن أى مساسٍ بهذا التقسيم سيُحوِّل العملية البرمجية إلى حلبة "جودو"، حيث تتداخل أقدام و أذرع المتصارِعين بشكلٍ لا يدرى معه أحدٌ لمن هذا الذراع و لمن هذه القدم !. و بالطبع فإن هذه الحال لا تُرضِى من يُدافِع عن الحق الطبعى للمُبرمِج العادى فى الحصول على الحرية القُصْوَى و الراحة الكاملة فى التركيز على حل مُشكلته التى يرغب فى حلها، بدلاً من تفريغ جزءٍ من وقته و جهده لعمليةٍ من الواضح أنه لا ناقة له فيها و لا جمل.
ثم إننى أتساءل عن الوظيفة التى نَدَّخرها للمُترجِم إن كانت إدارة الذاكرة من نصيب المبرمج العادى، فهل ستكون مهمة المُترجِمات الرئيسة التى تُرَكِّز عليها أغلب التركيز هى عملية التحويل من النص البرمجى عالى المستوى إلى الأصفار و الآحاد فقط و نترك المهمات الوعرة الأخري للمُبرمِج ليتولّاها بنفسه ؟

إننى أظن أنه من اللازم حقاً أن يُرَكِّز مُطوِّروا اللغات البرمجية على النُّموذج النظيف منها، و أن يضعوا نُصْب أعينهم تطوير مُترجِمات كفؤةٍ تُعِين المُبرمِج المُستخدِم على التركيز على عمله الأساسى، و فى نفس الوقت تُطبِّق أفضل الخُوارِزْمات التى تُعطِى البرنامج كفاءة الإستغلال الأفضل لموارد الجهاز.
إن جعل إدارة الذاكرة من نصيب المُبرمِج النهائى لا يكتفى فقط ببلبلة فِكْر المُبرمِج و شَغلِه عن الأمر الأساسى له (و هو برمجة التطبيق المرغوب فيه)، بل يَفرِض عليه أن يَعْلم أشياءاً لا تُهِمه فى قليلٍ أو كثيرٍ فى الأدوات البرمجية التى يَستخدمها.
و سأَضرِبُ لكم مثالاً شخصياً علي هذا الأمر:
فأنا عن نفسى كنتُ أستخدم مكتبة الـOpenCV الحرة المصدر لأنى كنتُ أعمل فى مركز أبحاثٍ فى مشروعٍ يتعلق فى شِقِّه الأول بمُعالَجة الصور الرقمية. و كنتُ أستخدم لغة الـC++ التى تجعلنى أُعانى الأَمرَّينِ؛ فالبرنامج الذى يحتاج منى إلى يوم لكتابة خُوارِزْمه و ضبطه منطقياً يحتاج على الأقل ليوم آخر لكتابة الكود نفسه !، مع أنه من المُفترَض أن مرحلة التكويد هى أبسط المراحل فى البرمجة، بل و يَحتاجُ إلى أكثر من يومٍ آخرٍ للتنقيح و التخلص من الأغلاط التى تظهر نتيجةً للقيود التى تضعها اللغة المُستخدَمة على كاهِلى (أناْ البائس المسكين).

و مثالٌ لذلك: البرنامج التالى الذى فيه غلطةٌ منطقيةٌ تظهر قبل انتهاء عمل البرنامج:

```
// step1: Load codebook binary file
Network net("D:\\codebook archive\\gray_codebook\\gray_codebook.bin", false);
// getting input from camera
CvCapture * camera= cvCreateCameraCapture(0);
// taking the current shot of the camera in an IplImage structure to work on
IplImage * colored_frame = cvQueryFrame(camera);
// making a new window to show the image in it
cvNamedWindow("camera view");
// keeping working on the frames taken from the camera untill the user wants some thing
while(true)
{
    // get the new frame
    colored_frame = cvQueryFrame(camera);
     |
     |
     |
     |
     |
     |
     |

    cvDestroyWindow("camera view");
    cvReleaseCapture(&camera);
    cvReleaseImage(&colored_frame);
```

ففى هذا البرنامج أقوم بالقراءة من الكاميرا الرقمية المُوَصّلة بالحاسوب و أُسجّل ما أقرأ فى كائنٍ يُمثّل لقطة كاميرا، بعد ذلك أقوم بأخذ اللقطة الجديدة من ذلك الملف إلي كائنٍ يُمثّل صورة، و من ثم أقوم بعرض هذه الصورة فى نافذةٍ على الشاشة، و بعد أن أنهى العمل عليها أقوم بإغلاق النافذة و تحرير الذاكرة المُخصّصة لملفى اللقطة و الصورة.

فما هى تلك الغلطة المنطقية التى تظهر ؟!

اتضَحَ لى فى النهاية أننى كَتبتُ الأمر الخاص بتحرير جزء الذاكرة المحجوز للصورة الجديدة من الكاميرا بعد أن حَرّرتُ جزء الذاكرة الخاص بملف لقطة الكاميرا،

فما ذا فى هذا ؟!

كانت المُشكِلة أن الـOpenCV حينما عرّفتُ صورةً جديدةً علي أنها مأخوذةً من ملف اللقطة (المقروء من الكاميرا) لم تنسخ الصورة من ملف اللقطة إلى مكانٍ جديدٍ فى الذاكرة لتعمل عليها، بل جعلت الكائن من النوع (صورة) يُشِير إلى الصورة و هى فى مكانها الأول فى الذاكرة الخاصة بملف لقطة الكاميرا.

إلى هنا و الأمور جيدة؛ فهذا يُفِيد فى الحصول على أداءٍ فائق عند التنفيذ، لكن المُشكِلة أن اللغة تَطلبُ منى تحرير الذاكرة التى أستخدمُها بنفسى، و حينما فَعلتُ لم أنتبه إلى كيفية تعامل المكتبة مع الذاكرة: فحررتُ اللقطة ثم عُدتُ و حررتُ الصورة التى تم تحريرها أصلاً من ضمن لقطة الكاميرا، أى أننى بهذا أكون قد حررتُ جزءاً من الذاكرة أصبح غير محجوزٍ أصلاً !

و السبب هو التواطؤ غير المقبول بين المكتبة و اللغة، و لو أن اللغة كانت مُتعاونةً لكانت هذه ميزةً فى مكتبة الـOpenCV و ما كنتُ لأشغل بالِى بكيفية التحسين الذى وضعه مُصمّموها و مُبرمِجوها ليجعلوها

تُعالِج الصور بشكلٍ أسرع، و كنتُ سأستفيد منه بدون أن أشغل ذهنى بما فعلوا بتاتاً، و لكن استخدام الـ++C جَعَل الأمر كابوساً بالفعل و فَرَض عَلَيّ أن أعلم كيف يُدِيرون الذاكرة حتى لا أقع فى أخطاءٍ منطقيةٍ نتيجةً لمضاداة الكود الذى أكتبُه للكود الذى يكتبونه هم !، أى أن مُبرمِج الـ++C مُعرّضٌ فـي أي لحظة لأن تُصبِح خاصية التجريد abstraction ليس لها أى معنى معه !

فإذا كان الأمر هنا هيّناً و يُمكِن التعايش معه فإنه سيُصبِح مأساةً حقيقيةً إذا ما نظرنا إلى المشاريع البرمجية الضخمة التى تُبرمَج باستخدام الـ++C، مثل أسطح المكتب الرسومية و أنوية أنظمة التشغيل و غيرها من المشاريع العملاقة، و التى يتم فيها بطبيعة الحال التعامل مع الذاكرة دَوْماً و فى كل مَنْحَىً من المَنَاحى بشكلٍ فى غاية التعقيد بسبب استخدام لغة الـC أو الـ++C فى التكويد، و سيكون عَلَيّ أن أُعانِى الأمَرّين لو أردتُ المشاركة فى مشروعٍ كهذا لمجرد فهم كيفية سير الأمور؛ حتى لا أسير فى إتجاهٍ مُعاكِسٍ فيحدث ما لا تُحمَد عُقْباه !
أى أننى بدلًا من أقضى الوقت فى التفكير فى كيفية تطوير المشروع و تحسينه و إضافة الأفضل إليه: سأقضى أغلب الوقت أتعلم أشياءاً هى من صَمِيم مُكوِّناتٍ أُخرى من المشروع، و لكنها تتعلق بالذاكرة التى عَلَيّ أن أُدِيرها حينما أتعامل مع ذلك المُكوَن أو غيره. و لو أننى أهملتُ هذه المسألة فسأدفع الثمن غالياً بالحصول على ذاكرةٍ كبيرةٍ مُهْدَرة، و هو الأمر الذى سيُؤدِّى إلى جَعْل فريق العمل يركل مؤخرتى و يطردني بلا رجعةٍ منه؛ نظراً لأن الأنظمة التى يَبنوها لابد من أن تكون فى غاية الكفاءة فى استخدام موارد النظام و لا يُسمَح فيها بإهدار أى جزءٍ من الموارد على الإطلاق.

و من الغريب أن الكثيرين من المُبرمِجين مُقتنعون اقتناعاً تاماً بأن وَضْع إدارة الذاكرة بين يدى المُبرمِج النهائى يُعطِى البرامج المُنْتَجة قوةً و كفاءةً عاليتين، و أصبح هذا القول بدوره من التابوهات المُحرّمة فى عالَم البرمجة، و أصبح الإقتراب منه يُعَرِّض المُشَكِّك للسخرية اللاذعة على أساس أنه لا يعلم أبسط القواعد فى عالم البرمجة الاحترافية. رغم أن دراسة (The Measured Cost of Conservative Garbage Collection by B. Zorn) التي أُجرِيَت للمُقارَنة بين كفاءة البرامج التى تُسَلِّم إدارة الذاكرة إلى المبرمج النهائى في لغة الـC و البرامج التى أَسْلَم مُبرمِجوها أمر إدارة الذاكرة و جمع المُهمَل منها إلى مُجمّع نفاياتٍ جيدٍ: أثبتت أن البرامج الثانية كانت أفضل من ناحية الأداء، على العكس تماما من القناعات التى رسَخها كثيرٌ من المبرمجين القُدامى فى أذهان الكل !

و دعونا نَفترض أن لكلا النوعين من البرامج نفس الأداء و مُعدَّل استغلال الذاكرة، بل فلنَفترض أن إدارة الذاكرة اليدوية تُعطِي نتائجاً أفضل: فماذا عن الأغلاط التي تملأ البرنامج و تُحِيل حياة المُبرمِجين إلي جحيم لا يُطاق، و تُحوِّل البرنامج إلي أرضٍ مَلأي بالألغام المُعرَّضة للإنفجار في أي لحظة ؟!
ماذا عن الوقت الضائع في التنقيح و البحث عن مواضع الأغلاط و كيفية حلها ؟!
ماذا عن تكلفة كل هذا ؟!

ما ذا عن الكود المُتضخِّم بلا معني، بينما أكواد البرامج التي تستعمل مُجمِّع نفاياتٍ أصغر و أَوْضَح ؟!

ما يُمكِنني أن أقوله بكل ثقةٍ أن قَناعة تَفوُّق الإدارة اليدوية للذاكرة علي الإدارة الآلية ليست إلا تجلياً إضافياً للإنعزال المعنوي عن التِقنِيات الحديثة فى عالم البرمجة لدى أجيالٍ من المُبرمِجين القُدامَى و المُحدَثين. تماماً كرفض استعمال الواجهات الرسومية و بيئات التطوير المُتكامِلة و لغات البرمجة الحديثة. و الإصرار على إستخدام سطر الأوامر **فى كل الأمور**.

و لو أننا التمسنا العُذر لقُدَامى المُبرمِجين فى ذلك (على أساس أنهم نشأوا على سطر الأوامر و تَعوَّدوا عليه، و كذلك أصبحوا خُبراء لا يُشَق لهم غبارٌ فى البرمجة باستخدام اللغات التى استخدموها منذ صِغَرهم، و تكيَّفوا مع عيوبها قبل مُميِّزاتها و أصبح فى غاية الصعوبة بالنسبة لهم البدء من الصفر مع لغةٍ جديدة)، أقول: لو التمسنا لهم العذر بسبب ما فات ذِكْره لما استطعنا التماس العُذر للأجيال الجديدة من المُبرمِجين الذين نشأوا و تربوا على التقنيات الأكثر حَداثةً من تلك التى اعتاد عليها قُدَامى المُبرمِجين (مِثلى بالضبط) و لكنهم استخدموا تلك الأقدم على الرغم من ذلك ؟!

فالسبب الوحيد الذى أراه دَفَعهم إلى هذا هو: الجانب النفسى الذى جعلهم يتصورون أنه ما دام القُدَامى مُصِرِّين على استخدام التقنيات و الأدوات القديمة: فهى الأفضل بلا جدال. و استخدامها دليلٌ على الإحتراف و التمكُّن، أما التقنيات الجديدة فلا يستخدمها إلا المُرَفَّهون من المُبرمِجين الصغار الذين لم يَصِلوا إلى مرتبة الإحتراف بعد و مازال الطريق أمامهم طويلاً !

و أنا بِطبعى أكره التعصب لتقنيةٍ على حساب تقنيةٍ أخرى لمُجرَّد الأُلفة مع التقنية الأولى و عدم اعتياد الثانية. أو لأن استخدام الأُولى يُوحِى بالخبرة العميقة فى البرمجة و القدرة على أداء الأعمال الغاية فى الصعوبة ككِبار المُبرمِجين مِمَّن يَستخدِمون التقنية الأولى دوماً. و يُعجِبنى دائماً المُبرمِج الذى يتعامل مع التقنيات المُختلِفة على أساس الإستفادة القُصْوى من كل واحدةٍ منها و الإبتعاد قَدْر الإمكان عن مساويها و عيوبها. و هذا هو قول العقل السليم الذى يبحث عن الأفضل فى كل شئ.

فنَجِد المُبرمِج العاقِل يستخدم سطر الأوامر فى أمورٍ مُعيَّنةٍ و يستخدم الواجهة الرسومية فى أمورٍ أخرى. و يُبرمِج بنفسه البرامج التى يحتاجها فى أمرٍ ثالث. و يستخدم برامج الآخرين فى أمورٍ أخرى. و فى كل مجالٍ من المجالات يحاول اختيار اللغة البرمجية الأفضل (إن استطاع تعديد اللغات التى يستخدمها) بدون تعصبٍ لتلك اللغة على حساب تلك.

الوِراثة:

حينما يتحدث أي شخصٍ عن مفهوم الوراثة فإنه يَقصد نقطة: صُنْع صِنْفٍ من الأصناف، ثم وراثة مُحتوَياته فى صِنْفٍ آخرٍ و التغيير فيها بالإضافة و/أو التعديل. أما مفهوم الوراثة عندى فهو أمرٌ يتجاوز ذلك إلى معنىً أوسع و أشمل؛ فيحتوى على أى طريقةٍ لإعادة استخدام الأكواد فى البرامج بأى شكلٍ من الأشكال بشرط أن يَضمن تأثر الكود الوارث بالتغييرات التى تطرأ بعد الوراثة على الكود الموروث، و بالطبعِ فإن هذا يستثنى طريقة النَّسْخ و اللصْق التقليدية؛ لأنه فيها تنقطع العلاقة بين الوارث و الموروث تماما بمُجرَّد انتهاء العملية.

لذلك فمفهوم الوراثة عندى يتضمن الأمور التالية:

- **الوراثة العادية** بين صِنْفٍ من الأصناف و صِنْفٍ آخر، و هو التعريف النمطى الذى يُعطِيه الكل للوراثة فى البرمجة الكائنية المَنْحَى oop.

- **تعريف كائنٍ أو نُسخَةٍ من صِنْف**؛ فلو فكَّرنا فيها جيداً لوَجَدنا أن الكائنات التى نُعلِن عنها ما هى إلا إعادة استخدامٍ لكود الصِنْف بدون الحاجة إلى كتابته مَرَّةً أخرى لكل كائنٍ كما يحدث فى منهجيتى البرمجة الوظائفية functional programming و الإجرائية procedural programming، و كذلك فإن أى تغييرٍ يطرأ على كود الصنف سيكون مُطبَّقاً على كافة النُسَخ المُنتَجة منه. و بذا يتحقق شَرْطَى الوراثة عندى على هذه العملية.

- **استخدام المكتبات البرمجية ذاته**. و هذا الأمر قد يَدفع إلى الظنِ بأن الوراثة عندى ما هى إلا مُسَمّىً آخر لعملية إعادة الإستخدام، و لكنه ظنٌ خاطئٌ تماماً؛ ففى عملية إعادة الإستخدام يكون الشرط الرئيس لكى يُمكنك أن تقول أن ما حدث هو إعادة استخدام هو أن يقوم المُبرمِج باستخدام كودٍ برمجيٍ مرةً أخرى فقط، أى أنه لا يُشترَط لكى تكون العملية ضمن نِطاق إعادة الاستخدام أن تكون التغييرات الجديدة فى الأصل (و التى تمَّت بعد تنفيذ عملية إعادة الاستخدام) ذات أثرٍ على الفروع المُستخدِمة للأصل البرمجى، فى حين أن هذا شرطٌ أساسىٌ عندى لكى أُطلِق على العملية لقب الوراثة.

و لو نظرنا إلى عملية النسْخ و اللصق التقليدية لوجدنا أنها خير مثالٍ يُوضِّح الفارق عندي بين الوراثة و إعادة الاستخدام؛ لأنها يَصدُق عليها وصف إعادة الاستخدام بكل قوة، فى حين أنه لا ينطبق عليها مفهوم الوراثة عندى و لا يُمكِننى أن أعُدّها لوناً من ألوان التوراث لعدم تطبيق التغيُّرات المُحدَثة إلا على الأصل المنسوخ فقط.

و قد أفضتُ فى شرح الفارق بين نظرتى للوراثة و بين النظرة الحالية لها لكى أستطيع بعد ذلك أن أتحدث بحريةٍ عن مدى أهمية الوراثة عندى، و ذلك بدون خوف الإلتباس على القارئ لهذه الأسطر.
و الحق أن للوراثة عندى أهميةٌ فى الحُدود القُصْوى لها؛ فأنا أعتقد اعتقاداً جازماً أن أى كودٍ مُهَنْدَسٍ جيداً قد يكون قابلًا لإعادة الاستخدام التوارثى بطريقة تُمكّننا من حل المشاكل الجديدة بطرقٍ أسرع و أفضل بكثير.

و جديرٌ بالذِّكر أن الوراثة تَدْخُل فى أغلب المواضيع الهامّة التالية إن لم يكن فيها كلها، و ذلك لأن لها من الفوائد الكثير، و يُمكِننى أن أَعُد هنا طرفاً من هذه الفوائد فيما يلى:

- **سرعة الإنتاج:**

و القول المُنتشِر الذى يَصِف هذه النقطة جيداً هو "عدم إعادة اختراع العَجَلة" (علي الرغم من إساءة تفسير البعض لهذا المبدأ)، حيث لن يكون على المُبرمِج كتابة الآلاف من الأسطر فى كل مرةٍ يحتاجها فى عمله؛ بل يكتبها مَرّةً واحدةً فقط ثم يقوم باستدعائها كل مرةٍ بمُنتَهى السرعة، مما يَعنى التركيز على المُشكِلة الحقيقية و عدم إهدار الجهد فى أمرٍ فُعِل من قَبْل و هو ما يُؤّدى إلى الإنتاجية الكبيرة فى النهاية.

- **الأمن:**

الأكواد التى ستُستَخدَم فى الوراثة ستكون قد اختُبِرَت جيداً قبل وَضْعِها فى الخدمة التَّوَارُثية، مما يَعنِى أن المُبرمِج المُستخدِم لهذه الأكواد الموروثة يستطيع الإطمئنان إلى كَوْن الكود الموروث ذو أمانٍ عال، فلا يَحتاج إلى إعادة اختباره مَرّةً أخرى، و ذلك على العكس من الأكواد الجديدة التى يكتبها إن لم يَستخدِم الوراثة؛ حيث سيكون عليه التأكد من أَمْن تلك الأكواد فى كل مَرّة.

- **تَوْفِير التكلفة:**

عند إنتاج البرمجيات بصورةٍ تجارية يكون من أهم الأمور: إنتاج التطبيق المُراد بأقل التكاليف المُمكِنة، و قد قام علم هندسة البرمجيات من الأصل ليَصِل إلى هذا الهدف من ضمن الأهداف الرئيسة التى قام لأجلها.

- **تَوارُث الكفاءة:**

حينما يتم اكتشاف خُوارِزْم جديدٍ أكثر كفاءةً للقيام بالعملية التى كان يُستخدَم فيها خُوارِزْمٌ أقل كفاءةً: يكون إحلال الجديد محل القديم مكسباً كبيراً؛ ليس فقط للتطبيقات التى ستَستخدِم الخُوارِزْم بعد الإحلال، بل حتى للتطبيقات التى تَستخدِم الخُوارِزْم من أيامه الأولى

قبل التجديد، أى أن التغييرات الجيّدة لن تجد طريقها فقط للمُستخدِمين الجدد بل سَيتِم توارثها هى الأخرى (بالطبع قد يحتاج هذا إلي إعادة ترجمة recompile الأكواد التي تَستخدِم الخُوارِزْم مَرّةً أخري).

- **التَّجرُّد الأقصى و بساطة حل المشاكل:**

حينما تتوافر تحت يد المُبرمِج أدواتٌ كثيرةٌ جاهزةٌ للاستخدام: سيكون الأمر الوحيد الذى سيُركّز عليه جهده و تفكيره هو حلِ المشكلة التى يُرِيد استخدام البرمجة لحلها، و هو الأمر الذى سيَجعل البرمجة أكثر سهولةً بكثير (بل و أكثر إمتاعاً بكثير).

التَّهْجِين:

يَتفاخر كثيرٌ من مُصمّمي لغات البرمجة المُختلِفة و المُستخدِمين المُتعصِّبين لها اليوم بأنها لغاتٌ كائنيةٌ أو وظائفيةٌ أو إجرائيةٌ صرفة، و ذلك من مُنطلَق أن الخير كل الخير فى ذلك النَّمَط البرمجى الذى تَخصّصت فيه تلك اللغات و لا خَيْر فيما عاداه، و هو القول الذى أود أن أنوه هنا إلي خطئه الفادح الواضح.
نرى من يتعصب للـjava يقول أن البرمجة الكائنية هى الأفضل، و هى النوع الأقوى من الأنماط البرمجية التى وَصَل إليها الفكر البرمجى البشرى، و بالتالى كان من الرائع أن تكون الـjava تامَّة الكائنية pure oop. و هناك من يُصِر على أن البرمجة الوظائفية أو الإجرائية هى الأفضل و الأقدر على القيام بكل المهام بسهولة، و بالتالى تكون البرمجة بِلُغاتٍ مثل الـC مُنتهَى المتعة بالنسبة له.

و الحق أننى لا أرى الحق مع أيٍ منهما، بل أظنهما رأيا ما امتلأ من الكُوب و لم يَنتبِها إلى ما فَرَغ فيه؛ فالمُبرمِج باللغة الكائنية الصرفة لم يَنتبِه إلى ما بها من عُيوبٍ كان بإمكانه تفاديها عند البرمجة بالنَّمط الوظائفى أو الإجرائى، و كذا فالمُبرمِج الوظائفى أو الإجرائى (إن صَحَّ لى أن أُسَمّيه هكذا) لم يَنتبِه إلى أنه كان بإمكانه الإفادة من ميزات البرمجة الكائنية بدلاً من المُعاناة التى وَضَع نفسه فيها حينما استخدم نَمَط البرمجة الآخر بمُفرده.
و قولى هنا يَستنِد إلى أن كل نَمَطٍ برمجيٍ له ميزاته و عيوبه التى يَعْلَمها كل من دَرَس علم هندسة البرمجيات و مَارَس البرمجة عملياً بشكلٍ كبير، و ما دام الأمر هكذا فإنه من الأفضل لو استطاع المُبرمِج أن يَستخدِم كل نَمَطٍ من هذه الأنْماط فى أفضل النَّواحى التى يَتميَّز فيها ذلك النَّمَط و يكون من الأكفأ استعماله فيه.

و ما دامت اللغة البرمجية هى الأداة التى يَستعملها المُبرمِج فى عمله فإن هذه الأداة ستكون هى الفَيْصَل فى تحديد الاختيارات التى سيَقُوم المُبرمِج باختيار نَمَط البرمجة المُناسِب من بينها، فلو كانت اللغة صرفة النَّمَط (سواءٌ أكانت كائنيةً أم وظائفيةً أم إجرائيةً) فإنها تَقومُ عندئذٍ بفَرْض ذلك النَّمَط على المُبرمِج و تُكَبِّله به و تُقيّده به تماماً فلا يُفارقه، أى أنها تُعطِيه كل ميزات ذلك النمط و لكنها تُحَمِّله كل عيوبه، و فى نفس الوقت تَحمِيه من عُيوب الأنماط الأُخرى و لكنها تَحرِمه من المُميِّزات التى تَتمتَّع بها و لا تُوجَد فى نَمَطِها المُختار.

و المُجادَلة بأنه يُمكِن ببعض الإلتفاف بناء نَمَطٍ باستخدام نَمَطٍ آخر هو نوعٌ من الإستِجارة بالنار من الرَّمْضاء؛ فما دام هناك الطريق الأسهل و هو إتاحة الفُرصة أمام المُبرمِج ليختار النَّمَط الذى يَرتَضِيه لتطبيقه: فلِمَ تكلُّف العَناء و إثقال عاتِق المُبرمِج بمهمة الإلتفاف و الدوران ليَحصُل على ما يُريد ؟!
ثم فى النهاية نَجِد أن لهذا الأمر أضراره التى لا تَخفَى على أحدٍ و منها:

- هناك أَنْماطٌ يكون من الصعب أن تُعلِّمَها للمُبرمِج المُبتَدئ فى البداية قبل الأَنْماط الأخرى، مثل البرمجة الكائنية التى لا يُمكِن لأحدٍ أن يَفهمها جيداً قبل أن يَفهم أولاً البرمجة الإجرائية و الوظائفية، و ما يَحدث عندما يكون أمامه لغة برمجةٍ كائنيةٍ صرفةٍ مثل الـjava و يبدأ تعليمه بها ببرنامجٍ هو الأبسط فى أى لغةٍ مثل:

```
public class Hello{
    public static void main(String [] args){
        System.out.println("Hello CCSC");
    }
}
```

ثم يَتساءل ببراءةٍ عن معانى public و class و static: أن مُعلِّمه يبدأ فى قول كلام كثيرٍ من عيّنة (لن نتحدث عن هذا الآن) و (اكتبها كما هى و سأُخبرك بالسبب فى وقتٍ لاحق) و غيره الكثير من الكلام الذى لا يجد فى نَفْس المُستمِع إلا دَوْر إثارة الحيرة و الإرتباك بشدة.

و يكون بعدها من الجَلِى أنه لا يُمكِننا تعليم البرمجة بلغةٍ مثلها لأننا نَحتاج إلى المرور على اللغات الأخرى أولاً قبل المَجِئ إليها، و بالتالى يَضيع الكثير من الوقت على المُبرمِج المُبتدئ مُتخبّطاً بين اللغات المختلفة لمُجرّد تعلُّم البرمجة بصورةٍ جيدة، و هو الأمر الذى لا أجِد له داعياً من أى نوع.

و خِلاف ذلك الوضْع مع اللغات التى لا تَدعم البرمجة الكائنية مثل الـC؛ حيث يكون على الراغب فى تعلُّم البرمجة الكائنية الذهاب إلى لغةٍ أخرى، و لعل ما يُخفِّف من عناء ذلك و مشقته فى حالة مُبرمِجى الـC أنه سيَجد لغة الـC++ التى يُمكِنه أن يُغمِض عينيه بعض الشئ عن الفُروقات التى بينها و بين الـC ليقول (بضميرٍ غير مُستريح) أنهما ذات اللغة، و أنه لم يُضطَّر إلى تَرْك اللغة الأولى لتكملة مشوار التعلُّم البرمجى. و لكن الحقيقة تظل هى الحقيقة من حيث أنه حتى فى هذه الحالة فإن الأمر كان مُرهِقاً بالنسبة له و لم يكن كما يُحاوِل إيهامنا نزهةً ريفيةً جميلة.

و حتى فى تلك الحالة يظل القول بأن المُبرمِج ما زال تحت سيطرة نوعٍ واحدٍ من الأَنْماط البرمجية حقيقةً لها مُنتهَى القوة و العُنْفُوان، و يظل الحصول على ميزات الأَنْماط البرمجية الأخرى مرهوناً بمدى البراعة و الدهاء و الوقت و الجهد، مما في إمكان المُبرمِج إبداءه و بَذْله للإلتفاف حول النَّمَط الوحيد للّغة لبناء النَّمَط المطلوب. و هكذا سنَرى إهدار الوقت و الجهد و حِيَل الحُواة و المُشعوِذين فى أكواد المُبرمِجين من تلك النوعيات المسكينة المقهورة على أمرها.

لذلك و بكل بَساطةٍ أستطيع أن أقول أن المبدأ الذى يجب على كل لغة برمجةٍ أن تنتهجه عند التعامل مع أَنْماط البرمجة المُختلِفة هو: التّهْجِين، و أقصد بالتّهْجِين هنا مَزْج أقصى قدرٍ مُمكِنٍ من الأَنْماط البرمجية فى اللغة بحيث يكون المُبرمِج قادراً على استخدام أيها حينما يُرِيد ذلك، مدفوعاً بالطبع إلى ذلك بقَراراتٍ تصميميةٍ تُخبِره أن ذلك هو القرار التصميمى الأفضل فى هذه الحالة.

و ليس الأمر بالصعوبة التى سيتصورها البعض؛ و الحق أن إلقاء نظرةٍ واحدةٍ على لغة إبداع و قواعد تصميمها يكفى لشرح الأمر بكل وُضوح بدون الحاجة إلى مَزيد قَوْل؛ ففى إبداع يُمكِن للمُبرمِج أن يُبرمِج بالنَّمَط البرمجى الذى يُريده سواءٌ أكان نَمَطاً إجرائياً (أو البرمجة بالأعراض الجانبية side effects كما يُسمُّونها أحياناً) أم كان نَمَطاً وظائفياً (و هو يُضاد الأعراض الجانبية تماماً) أو حتى النَّمَط الكائنى الواضح القوة، و كل هذا لم يُعَقِّد تصميم اللغة كما قد يَتصور البعض، و لكنه على العكس كان له سِمة التسهيل و التخفيف الشديدين (على الأقل بالنسبة لى).

الشّمُول:

حينما تُريد برمجة تطبيقٍ نِظامي يَحتاج إلى الأداء الأفضل: فعليك بالبرمجة بلغة الـC. و حينما تُريد برنامجاً يَتعامل مع النوافِذ و الواجِهات المَرئية أو قَواعِد البَيانات فعليك بالـjava أو الـvisual basic أو الـ#C. و ... و ...

ما فات هو كلامٌ أَثِقُ أن الكثيرين (إن لم تَكن الأغلبية) قد سَمِعوه مِراراً و تِكراراً فى المُنْتَديات المُختلِفة و الكتب المُتنوّعة، التى تُناقِش موضوع الترجيح و المُفاضَلة بين لغات البرمجة المُختلِفة. و لا تَكون الفائدة التى يَخرج بها القارئ بعد ذلك إلا الإحساس أن كل تلك اللغات لا يُغنِى أحدها عن الآخر، و لا يُمثِّل واحداً منها دَوْر الخادم الجيّد فى كل مجال، و هو الإحساس الذى لا يُسعِد أحداً بحالٍ من الأحوال.

و أنا أَجِد أنه من حقى كمُبرمِجٍ بائس أن أَقف لأتساءل عن السبب فى هذا، فهل السبب أن كل مجالٍ من مجالات الحاسوب البرمجية يَختلف عن المجالات الأخرى بدرجةٍ تَجعل اللغة التى تُناسِبه برمجياً لا تُناسِب الأخري بنفس القدر ؟

أم أن السبب أن كل لغةٍ كان لها مُجتمَعها التقنى الذى رَكَّز على جانبٍ من الجوانب البرمجية، و تَراخَى عن تغطية المناطق الأخرى بِذات القوة، و من ثم اشتُهِرت اللغة بمجال التركيز الأكبر و أُهمِلت فى الجوانب الأخرى ؟

أم قد يكون السبب أن كل مجالٍ من المجالات كان له أهله ممن قام على أكتافهم مُعظَم (أو كل) بنائه، و كانوا قَدَراً يستخدمون لغة موحدةً أو لغاتٍ قليلة العدد فى ذلك البناء، و بالتالى كانت تلك اللغة (أو تلك اللغات) هى الوحيدة أو الأكثر ظهوراً عند أى حديثٍ برمجيٍ عن ذلك الجانب، و رُوَيْداً رُوَيْداً أَصْبَحَتْ كأنها اللغة (أو اللغات) المُمَثِّلة له فعلياً ؟

أم قد يكون السبب الرئيس هو كل تلك الأسباب مُجتمِعة ؟

الحق الذى أراه أن كل تلك الأسباب لها واقعٌ يُحقِّقها و تَنطبق عليه فى تاريخ لغات البرمجة بدرجات قوةٍ تختلف من سببٍ لآخر؛ فالسبب الأول الذى يقول أن هناك مجالاتٍ برمجيةٍ تختلف كثيراً عن باقى المجالات (و بالتالى فإن لغات البرمجة ذات النوعيات التى تُناسِب تلك المجالات لا تُناسِب المجالات البرمجية الأخرى) يَتحقق فى أحوالٍ عديدةٍ منها ما يلى:

- **أنظمة الذكاء الاصطناعى:**

التى تَحتاج إلى استخدام القواعد المنطقية فى عملها، فمثل هذه الأنظمة لها مُستوىً عالٍ للغاية من التجريد و البُعد عن الآلات و طُرُق عملها، و بالتالى فإن لغة البرمجة التى تُستخدَم فيها لابد أن

تَكون بعيدةً كل البُعد عن التعقيد، بل يَجب أن تَكون سهلةً مُتدفِّقةً غزيرة الإنتاجية و تَستخدِم المنطق، مثل لغات الـprolog و الـlogo و غيرهن من نوعية تلك اللغات المنطقية العالية للغاية فى المُستوَى.

و لا يُمكِن مع مجالاتٍ كهذه أن نَستخدِم لغاتٍ مُنخفِضة المُستوَى مثل الـC؛ فكل ما سنجنيه حينها هو الإحباط و اليأس فى اليقظة، و الكوابيس و بلل الفراش في المنام.

- **أنظمة قواعد البيانات و التعامل معها:**

حيث نَجِد أنه يتم استخدام نوعٍ خاصٍ من اللغات البرمجية (مهما اختلف شكله فهو متشابهٌ) و يُسَمَّى "لغات الإستعلام المُهَيَّكَلة structured query languages" للتعامل معها، مثل الـSQL القياسى و oracle SQL و MS SQL و غيرهن، و التقنية الجديدة التى ابتكروها فى microsoft و هى تقنية الـLINQ التى يُستَهدَف أن تُستَخدَم للتعامل مع كل أنواع مصادر البيانات و ليس مجرد قواعد البيانات فقط.

أى أنه لم يتم استخدام لغات البرمجة العامة العادية مثل الـjava و الـC# على الرغم من أنه يُمكِننا التحايل لذلك و صُنْع مكتباتٍ لتلك اللغات تقوم بعمل الصِياغات المُختلِفة للغات الإستعلام (و إن كان ذلك صعباً جداً)، لكن الأمر ليس فى إمكانية فِعْل ذلك من عَدَمِه إنما يَكْمُن سر الموضوع فى طبيعة التعامل مع قواعد البيانات عامة.

و لذلك نَجِد أنهم لم يَقوموا بدمج إمكانيات لغات الاستعلام فى اللغات البرمجية العامة مُباشَرة، بل قاموا بإعطائها القدرة على استخدام أوامر لغات الاستعلام الصريحة، و إن كان هناك بعض التغليف اللازم لقبول تلك العبارات فى بيئة لغة البرمجة العادية، و المِثال التالى بلغة الـC# يُرِينا مثل ذلك التغليف عملياً:

```
SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
mySqlCommand.CommandText =
  "SELECT CustomerID AS MyCustomer, CompanyName, Address " +
  "FROM Customers AS Cust " +
  "WHERE CustomerID = 'ALFKI'";
SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
mySqlDataAdapter.SelectCommand = mySqlCommand;
```

على الرغم من أنه كان من المُمكِن الاستغناء عن كل ذلك بصُنْع إجراءٍ يَأخذ كمُعامِلاتٍ:

- أسماء الحقول التى سيتم اختيارها للتعامل معها،
- اسم الجدول الذى يَحويها،
- الشرط الذى يجب أن يَنطبق على القِيم المُختارة.

و هذا كما قلنا صعبٌ للغاية، و لكن الأمر ليس فى الصعوبة فقط إنما فى الإختلاف الشديد عن البرمجة العا دية و هو الأمر الذى تُعزى إليه هذه الصعوبة نفسها و تُعَد جانباً من جوانب تجلياته.

- **تطبيقات الواجهات المرئية:**

التى هى من أ سهل ما يُمكِن عند التعامل معها با ستخدام لغات البرمجة الكائنية، و أَ صْعَب ما يُمكِن لو تم التعامل معها بِلُغات البرمجة الإجرائية أو الوظائفية؛ حيث فى اللغات الكائنية يُمكِننى ببساطةٍ أن أقول (بمُفرَ دات اللغة البرمجية) أننى أُريد كائناً من نوع النافذة، و هذا الكائن يَحوى زراً له القيمة الفلانية، و صندوقَ نصٍ له القيمة العلانية و...و... إلى آخره، أى أن الأمر يَسيرٌ جداً و منطقيٌ للغاية.

فما ذا عن اللغات الوظائفية أو الإجرائية و تعاملها (بطريقةٍ وظائفيةٍ أو إجرائيةٍ صرفة) مع هذا الأمر ؟

أنا شخصياً قد تَعاملتُ مع مكتبة الـopenCV من خلال عملى فى مجال مُعالَجة الصور الرقمية digital image processing، و بالطبع كنتُ أتعامل معها با ستخدام لغة الـC++ للحصول على أَفضل أ داءٍ ممكنٍ (و لأ سبابٍ أخرى ليس هذا محلها)، و يُسعدنى أن أقول أن الأمر كان مأ ساةً بالنسبة لى.

فمثلًا كان عَلَيّ أن أ ستدعى الدالة (cvNamedWindow) التي لها الترويسة header التالية:

```
int cvNamedWindow(
    const char* name,
    int         flags = CV_WINDOW_AUTOSIZE
);
```

لكى تَقوم بإنشاء نافذةٍ مرئية، ثم كان عَلَيّ أن أ ستدعى دالةً أخري لكى أقوم مثلًا بضم عمو د تَعقب track bar إلى تلك النافذة التى صنعتُها توا، و هى دالة (cvCreateTrackbar) التى لها الترويسة التالية:

```
int cvCreateTrackbar(
  const char*        trackbar_name,
  const char*        window_name,
  int*               value,
  int                count,
  CvTrackbarCallback on_change
);
```

و لكى أَقوم بتكرار الأمر (أى إنشاء أكثر من نافذةٍ و ضم عمود تعقبٍ لكل نافذةٍ) يكون عَلَيّ أن أَقوم بتكرار الأوامر ذاتها مرارا؛ لأن خاصية كتابة صِنْفٍ يَفعل ذلك ثم الإشتقاق منه فى كل مرةٍ غير موجودةٍ (بالطبع) فى النَّمَطين الوظائفي و الإجرائي، بل هى (و ياللعجب) الفكرة الرئيسة فى البرمجة الكائنية.

● **بِناء أنظمة البِنْيَة التحتية البرمجية للحاسوب** مثل أنظمة التشغيل و أدوات البرمجة المُختلِفة من مُترجِماتٍ و مُفسِراتٍ و مُجمِعاتٍ و غيرهن؛ ففى هذا المجال نجد أن لغة الـC هى اللغة الأشهر و الأكثر تغلغلا و لا يُنافِسها إلا نسختيها الكائنيتين الـ++C و الـobjective c (و ربما قليلٌ جداً من اللغات الأخري)؛ بسبب أنها حينما صُنِعت منذ البداية كانت قد صُمِّمت لتكون لغة بناء للبنَى التحتية فى الأصل، وبذلك وضعوا فيها كل المُوا صَفات اللازمة لهذا فأَصْبَحَتْ انعكاساً قوياً للغاية للفِكر البرمجى فى تلك الناحية (و الإنعكاس هنا كان بالطبع حسب طريقة تفكير مُصمِّميها و ليس هو بالإنعكاس الوحيد المُمكِن).
و قد استسلم أغلب المُبرمِجين لتلك الحقيقة و لم يُحاوِل إلا القليلون منهم بحالٍ من الأحوال تغييرها بالإستغناء عن لغة الـC التى كَرِهها الكثيرون، و بالتالى كان من الطبعى أن اللغات الجديدة التى يتم تصميمها سوف يتم إهمال تحقيق المُوا صَفات اللازمة للتعامل مع البِنَى التحتية للحاسب فيها؛ لأنه لا حاجة لهذا مع الرضا باحتلال الـC لعرش ذلك المجال، و الأكثر طبعيةً هو الاهتمام بإعطاء اللغة الجديدة الميزات التى تُؤهِّلها للتخصص فى مجالٍ آخر يُمكِنها أن تُنافِس فيه بقوة.

الأمثلة السابقة كلها كانت لتوضيح أن هناك بالفعل مجالاتٍ برمجيةٍ تختلف عن المجالات الأخرى، بصورةٍ تَجعل اللغة البرمجية التى تُناسِب التعامل معها بقوةٍ لا تُناسِب العمل فى المجالات الأخرى، إلا بتعسفٍ لا يُؤدى إلا إلى بطء العمل و عدم كفاءته (هذا إن نجح من الأصل).
لكن هناك مجالاتٌ أظن أنها تختلف عن المجالات الأخرى فى طريقة التعامل معها برمجيا، و بالتالى صارت لها لغاتها البرمجية الخاصة، بل صار للغاتها نفسها سمتاً خاصاً لا تَخرُج عنه، و منها مجال برمجة مواقع الشبكة الذى يكاد يكون من المعلوم من العقل بالضرورة عند كل المُبرمِجين أنه مجالٌ له صفته الخاصة التى لا تُشبه صفات المجالات الأخرى، و مع الزمن و تَوَطُّن هذه الفكرة فى عقول المُبرمِجين منذ تعلمهم البرمجة أصبح الأمر بِدَوره تابوها من التابوهات المُحَرَّمة فى المُجتَمع البرمجى و لم يتعرض له أحدٌ بالتغيير الجرئ إلا مُؤخَّراً.
و أنا أريد فى هذه النقطة أن أؤكد أن التفرقةَ بين برمجة البرامج العادية و برمجة مواقع الشبكة، و القولَ بأن للّغات التى تُؤدِّى كل عملٍ من الأعمال أشكال تصميم مُختلِفة: أمرٌ خاطئ، و التفرقة هنا هى تفرقة زورٍ لا تَجوز فى شرع العقل؛ فالإختلاف الوحيد فى برمجة البرامج العادية و لنُسَمِّها (برامج سطح المكتب) و برمجة مواقع الشبكة يقع فى الصيغة النهائية التى سيخرج بها البرنامج.

من حيث كَوْنِها صِيغةً تستهدف المُعالِج مُبا شرةً أم تَستهدف مُفَسِراً: فمواقع الشبكة تعتمد على برامج المُتصفِّحات browsers فى عملها مثل Firefox و internet explorer و غيرهما من المُتصفِّحات دائماً، و لا تعمل بأى حالٍ من الأحوال بصورةٍ مُنفردةٍ مُستقلة، فى حين أن برامج سطح المكتب لها الخيار فى أن تكون مُعتمدةً على مُفسِرٍ مُعيَّنٍ مثل برمجيات الجافا التى تعمل على منصة الـJVM و برامج الـ#C و الـvisual basic.NET اللواتى يعملن على منصة الـNET. ، و لها كذلك الخيار فى أن تكون مُستقلةً تعمل مُبا شرةً على المُعالِج و تتحاور مع نظام التشغيل لتطلب منه ما تُريد من مهامٍ بنفسها لا من خلال و سيط.

و بالتالي نري أن التفرقة بالفعل غريبةٌ للغاية في تصميم لغات البرمجة؛ و قد قلتُ من قبل أننى لا أجد أى فارقٍ حقيقى من وجهة نظر المُبرمِج عالى المستوى، و دليلٌ على ذلك لغاتٌ كالـjava script التى لا تُشبِه لغة الـHTML من قريبٍ أو بعيد. و تُستخدَم فى برمجة مواقع الشبكة ضمن أكواد الـHTML، و لما ذا أضرب المثال بالـjava script فى حين أن الحبيبة الغالية الـjava موجودةٌ فى مجال تصميم مواقع الشبكة بكل عنفوانها و هى اللغة التى تُستَخدم فى الأ صل لبرمجة برامج سطح المكتب، و أنا هنا آخذ الـjava كدليلٍ أقوى و أكثر حسماً فى هذه النقطة لأنها تقول كلامى عن الافتعال فى التفرقة بين نوعى لغات البرمجة (لسطح المكتب و للشبكة) بصورةٍ عمليةٍ لا جِدال فيها.

بل إن هناك من حَطَّم هذا الفارق بشكلٍ وا سع، حيث قاموا في شركة قوقل بجعل من المُمكِن تحويل برامج سطح المكتب ذات الأكوا د الأ صلية native إلي برمجياتٍ تعمل علي الشبكة من خلال Google Native Client. أي أنه يكفيك كتابة برنامجك بأي لغة برمجةٍ عا دية، ثم ترجمته إلي لغة الآلة الخا صة بالمُعالِجين Intel x86 و ARM: و سيكون بإمكانك تشغيله بصورةٍ عا ديةٍ ليعمل علي المُتصفَّحات إذا وافَقَ بعض الشروط الأ سا سية.

و يُمكِنكم مُشاهدة مُحاضَرة (beyond javascript: programming the web with native client) علي قناة (google IO) و التي يَتحدث فيها (david springer) بالتفصيل عن الموضوع و يُقدِّم مثالاً عملياً عليه،

هذا عن السبب الأول، أما عن السبب الثانى و هو (أن كل مجالٍ من المجالات كان له أهله ممن قام على أكتافهم مَعظَم بنائه و كانوا قَدَرًا يستخدمون لغةً مُوَحَدةً أو لغاتٍ قليلة العد د فى ذلك البناء، و بالتالى كانت تلك اللغة أو تلك اللغات هى الوحيدة أو الأكثر ظهوراً عند أى حديثٍ برمجي عن ذلك الجانب، و رُوَيْداً رُوَيْداً أَصْبَحَتْ كأنها اللغة أو اللغات المُمَثِّلة له فعلياً) فهو بِدَوْرِه له من الحقيقة نصيبه، و الواقع يَدُل على ذلك.

حيث لو أخذنا على سبيل المثال بناء بيئات سطح المكتب فى أنظمة التشغيل المُختلِفة مثل KDE و GNOME و LXDE فى الأنظمة شبيهة اليُونكس unix-like، و غيرهن من البيئات المشهورة الأخرى: لوَجَدُنا أنها يَغلب عليها أنها قد بُنِيَت با ستخدام لغة الـ++C التى يتحقق فيها شرط اللغات التى

تصلح لهذا المجال، و لكن على الرغم من ذلك فقد كان من المُمكِن أن يتم بناؤها بلُغاتٍ أخرى غير الـ++C مثل الـjava أو الـobject pascal مثلًا، إلا أن ذلك لم ينتشر لأن المُبرمِجين الذين قاموا بذلك العمل كانوا مِمّن يَستخدِمون لغة الـ++C بحُكم كَوْنهم من المُهتمين بتطوير و بناء النُظُم الأساسية أو البنية التحتية فى عالم الحاسوب، و بالتالى أصبَح استخدام الـ++C فى هذا المجال أمراً طبعياً، بل و أصبح مع الزمن من المُقدّسات المُحَرَّم لمسها فى عالم البرمجة.

و على الرغم من كَوْن استخدام لغاتٍ أخرى فى هذه المجالات قد يكون أكثر نفعاً من حيث: السهولة فى البِناء، و الكِبَر فى الإنتاجية، و البساطة فى المُتابَعة و التطوير (و هى الأمور التى تسعى خلفها كل دراسات هندسة البرمجيات)، إلا أن الأمر قد صار مُقدّساً بالفعل عند المُبرمِجين القُدامى و المُحدَثين، و صار المساس به و لو من بعيدٍ مُثيراً لعواصفٍ من النقد الهَدّام لصاحب التعليق و لأفكاره، و أظن أنه لو كان مُجتمَع المُبرمِجين النِظاميين وَثنياً: لعبدوا آلهةً يُخصِّصونه للـC و الـ++C و لقَدّموا لها قرابيناً بشريةً ممن يكفرون بهن.

و هذه هى خطورة كَوْن المُهتمّين بمجالٍ مُعيّنٍ من المجالات الحاسوبية من مُستخدِمى لغة برمجةٍ مُفرَدة؛ فرُوَيْداً رُوَيْداً سوف تُسيطِر تلك اللغة على ذلك المجال حتى و إن لم تَكن خصائصها الذاتية مُناسِبةً له تماماً، و سيكون أمراً منطقياً أن تصير تلك اللغة البرمجية هى المُمَثِّل الرسمى الوحيد لذلك المجال، و أن تَصير باقى اللغات كالأيتام على موائد اللئام عند وُلوج مُطوّريها إلى عالمه، فلا كرامة لها على الرغم من كَوْن بعضها أنسب للمجال من تلك اللغة التى تُقْرَن به !

و السبب الأخير و هو أن (كل لغةٍ كان لها مُجتمَعها التقنى الذى رَكّز على جانبٍ من الجوانب البرمجية، و تَراخى عن الإهتمام بالمناطق الأخرى فلم يُغَطِّها بذات القوة، و من ثم اشتهرت اللغة بمجال التركيز الأكبر و أُهمِلَت فى الجوانب الأخرى) هو بدوره له نصيبه من الصحة.
و على سبيل المثال لا الحصر يُمكِننا أن نرى أن حال لغة الـvisual basic فى مجال التكامل مع نظام الويندوز windows و حِزمَة برامج مكتب ميكروسوفت Microsoft office جَعَلَها هى اللغة الأشهر فى هذا، و لأن microsoft رَكَّزت على هذا الموضوع بقوةٍ فقد كان من المُعتَاد أن نَسمع فى فترة عُنْفُوان الـvisual basic عِباراتٍ على شاكِلة (الإنتاجية العالية = visual basic) و (تطبيقٌ به نوافذٌ مرئيةٌ = تطبيقٌ مكتوبٌ بـvisual basic) على الرغم من أن هناك مجالاتٍ أخرى تَصْلُح لها الـvisual basic برمجياً مثل كل اللغات عالية المستوى العامة الاستخدام إلا أنها اشتهرت بهذا بالذات.

و لعل هناك سببٌ آخرٌ يَندرِج تحت القول بالتركيز على مجالٍ مُعيّنٍ هو: الإهمال المُتعمّد؛ ففى بعض الأحيان يكون هناك إهمالٌ مُتعمَّدٌ من مُنتِج اللغة البرمجية لمجالٍ من المجالات لغرضٍ ما (هو فى الغالب

تجارى أو يَصب فى خانة هدفٍ تجارى)، و لغة الـvisual basic و لغات الـNET. عموماً دليلٌ على ذلك كل الدلالة؛ فالمُبرمِج الذى يَستخدِم اللغات السابقة يكا د يكون قد حَكَم على نفسه بالبقاء تحت مِظلة أنظمة تشغيل microsoft للأبد. لأن المَحْمُولِية portability لم تكن أبداً فى أى يومٍ من الأيام هدفاً من أهدافها.

و لعل الشئ الذى يُخفِّف ذلك على مُبرمِجى الـNET. هو وُجود المشاريع مفتوحة المصدر التى تُغنَى عن بديلها المُنتَج من قِبَل microsoft و أشهرها (و ربما الوحيد) هو مشروع الـmono، الذى يتكون من بيئةٍ برمجيةٍ مُتكاملةٍ IDE يمكن ا ستعمالهما مع لغتي الـ#C و الـvisual basic، و بيئة العملِ التي تَحِل محل بيئة الـNET. الخا صة بزمن التشغيل على نظام الويندوز و تعمل على باقة أنظمة التشغيل المشهورة مثل اللينوكس و السولاريز و ماك (و التى تم إهمالها بطبيعة الحال فى بيئة الـNET. الخا صة بـmicrosoft).

و لكني أظن أنه حتى فى هذه الحالة فإن هناك عوائقاً ستظهر فى طريق من يَرغب فى الإ ستفا دة القُصْوى من اللغة؛ نظراً لأنه سيضطر إلى الحصول على التطويرات الجديدة فى بيئة العمل و اللغة مُتأخراً عن المبرمجين الذين يتعاملون مع microsoft نفسها. فاعتما د التغييرات الجديدة فى mono لن يحدث بين يومٍ و ليلة، بل سيأخذ وقتاً بعد التصريح عن تلك التغييرات ر سمياً (و بالطبع سيكون قد تم اعتما دها فى برمجِيات microsoft لتحصل على السبق فى التسويق كما هى العا دة)، اللهم إلا إ ذا تعاونت مايكرو سوفت مع فريق عمل mono بقوةٍ بحيث يُساير فريق عملها في الجدول الزمني.

و قد أ دى وجو د براءات اختراعاتٍ لمايكرو سوفت في بنية عمل مشروع mono إلى تحذير (ريتشار د ماثيو ستولمان) من ا ستخدامه في عمل البرمجيات مفتوحة المصدر؛ لأن مايكرو سوفت قا درةٌ في أي لحظةٍ بهذا الشكل علي هدم الأمر كله أو علي الأقل عرقلته بشدة. و أنا عن نفسى أقبل اعتراضه هذا تماما رغم أننى كنتُ من مُحترفِى البرمجة بالـ#C إلا أن الحق أحق أن يتبع. و إ سنا د الظهر إلى microsoft سيكون أكثر قرارات عمرى حماقةً إن اتخذتُه. لذا فأنا أ سأل الله تعالى ألا أضطر إليه فى يوم من الأيام.

***

كل ما فات كان توضيحاً لأ سباب عدم شمولية لغات البرمجة، و تخصصها فى مناطقٍ مُعينةٍ من مجالات الإبداع البرمجى، و ضعف أو غياب وجو دها فى باقى المجالات، إلا أننى لم أُوضِّح حتى الآن مَوقفى من كل ذلك الذى ذكرناه، و مَوقفى واضحٌ و بسيطٌ للغاية، و يُمكِن تلخيصه بدون أى إخلالٍ فى جُملةٍ واحدةٍ هى **(لغة البرمجة أداة المبرمج، و الأداة لا تكون أداةً جيدةً إلا إذا أغنت عن مثيلاتها فى مُعظَم العملِ إن لم يكن كله، فإن لم تَكن كذلك فإنها لا تستحق الإستعمال و ينبغى طَرْحها جانبا).**

أى أن اللغات التى تَشترَط على المُبرمِج التضحية بكثيرٍ من المجالات ليُمكنه التركيز على مجالٍ مُعَيّنٍ هى أ داةٌ سيئةٌ يجب على المُبرمِج نَبْذها؛ حتى لا تُكبِله و تَحُد من قدراته و أنشطته. أما لغة البرمجة الحقيقية التى تَستحق الإهتمام فهى التى تُعطِى المُبرمِج القدرة على الإبحار با ستخدامها فى أى مجالٍ برمجى يَعِن له أن يَضع قدمه فيه، أو على الأقل مُعظم المجالات البرمجية.

و بالتطبيق على لغات البرمجة الحالية سنرى أن اللغات التى تُحقّق هذا الشرط قليلةٌ للغاية مُقارَنةً بما هو مُستخدَم منها بالفعل، و أشهر الأمثلة التى ستُضرَب لهذا النوع من اللغات هى لغات الـ(java, C#, C++, visual basic .NET) لذا ففى تالِي كلماتى سوف أناقش موقفى من كل لغةٍ مُوضِّحاً صحة هذا الزعم من كذبه.

و أول تلك اللغات هى الـjava و و هى مِن أَحَبّ لغات البرمجة الحالية إلى قلبى؛ لكونها يَتحقق فيها مِن ميزات لغات البرمجة الجيدة عندى الكثير، و على الرغم من هذا فإن عندى انتقاداتٍ حادة للـjava فى بعضِ المسائل التصميمية (سأُصَرّح ببعضها فى أماكنٍ أخريٍ من هذا الكتاب، و سأُفْرِد للبعض الآخَر أوراقاً علميةَ خاصةً به)، لكن الأكثر أهميةً من هذا هو كَوْن الـjava قد حقّقت ميزة الشمولية هذه إلى حدودٍ كبيرةٍ للغاية، بما يَجعلها بحقٍ أشمل اللغات البرمجية التى أعرفها.

فالـjava تُعطِى المُبرمِج القدرة على البرمجة لمجالاتٍ واسعة المدى مثل:

1. برامج سطح المكتب ذات الواجهات الرسومية GUI applications،
2. برامج سطح المكتب النصية console applications،
3. برامج العملاء على الشبكة client applications،
4. برامج الخادمات servers applications،
5. برمجة الأجهزة الذكية مثل الهواتف المحمولة و الثلّاجات المُعَدّلة و غيرهن من تلك النوعية من الأجهزة،
6. برمجة الألعاب،
7. التعامل مع قواعد البيانات.
8. و قد استخدمها البعض بالفعل في بناء أنظمة تشغيلٍ مثل Jx و Jnode و ربما أخريات !

و بالتالى فليس هناك مجالٌ يُمكِن أن يُفكِّر فيه المبرمج إلا وَجَد الـjava واقفةً فى انتظاره تمدُّ إليه يد المساعدة، و أهم من ذلك كون الـjava تعمل على أنظمة التشغيل المختلفة بمُنتهى المُساواة و بنفس الكود البرمجى، و كما يُعبّر مُنتِجو الـjava عن هذا المبدأ بقولهم (اكتب مرةً واحدةً و شَغّل فى أى مكان) و يستحق هذا المبدأ كل ما يُبذَل من جهدٍ للوصول إلى تحقيقه.

أما الـC# فهى لغةٌ قويةٌ بالفعل و أراها أقوي من الـjava لأسبابٍ كثيرة، و لكن و رغم كل هذا فإن الـC# لا تصمد أمام الـjava إذا ما قارنّاهما من مُنطلَق الشمولية؛ و ذلك على الرغم من أنها تصلح لنفس المجالات التى ذكرناها للـjava ! و لكن بالطبع مع وَضْع جُمْلة (مع أنظمة تشغيل microsoft فقط) أمام اسم كل مجالٍ مما ذُكِر بالقائمة فى الأعلى، و هذا الأمر ليس بالهيّن أبداً إذا ما علمنا أن نسبة الخوادم servers التى تَستخدِم نظام تشغيل القنو/لينوكس تَبلغ ما يَزيد على الستين

فى المائة من مجموع الخوادم العاملة فعلًا، و أن أشهر المواقع العالمية تَستخدمه بدلًا من الـ windows.

و كذا حينما نَعلم أن هناك الكثيرين ممن يَتحولون سنوياً إلى أنظمة التشغيل الأخرى (و على الأخص القنو/لينوكس) بسبب ضَعف الـwindows الأمنى الشديد و تكلفته و تكلفة برامجه المُبالَغ فيها، بالتالى فإن القول بأن كل البرامج التى سيكتبها المُبرمِج لن تعمل إلا على أنظمة microsoft فقط هو ضربةٌ فى الصميم لكل ما حازته الـC# من شمولية، بل و ضربةٌ قاضيةٌ تماماً لأن المَحْمُولَية هى واحدةٌ من أهم أركان الشمولية، إن لم تكن بالفعل هى الرُّكْن الأهم و الأشد خطراً؛

فشموليةٌ محصورةٌ فى رُكْنٍ مُعينٍ من عالَم الحوسبة بالمُقارَنة بالشمولية المُنفتِحة على العالَم الحاسوبي كله تشبه بالضبط موقف الجارية من الزوجة (رغم الغرابة فى التشبيه)، فالزوجة و الجارية مُتساوِيتان فى الحق فى المُعامَلة الحسنة من الزوج أو المالك، و للزوج عليهما حقوقٌ مشتركةٌ كثيرة،

و لكن هل هناك أى مجالٍ للمُقارَنة بين وضع الزوجة و وضع الجارية ؟

أى صاحب عقلٍ سيرفض المُقارنة من الأصل، و بالمثل المُقارَنة بين شمولية الـjava و شمولية الـ C#: فهما إسمياً تكادان تتساويان تماماً من ناحية أنواع التطبيقات و المجالات التى يمكن أن يُستخدَما فيها، و لكن ذلك لا يكفى أبداً للمُساواة بين الزوجة (الـjava) و الجارية الـ(C#).

و ما قِيل عن الـC# يُقال بالضبط عن الـvisual basic .NET نظراً للتَّوْأمة التى صنعتها Microsoft بين اللغتين حينما ضمتهما إلى إطار عمل الـNET.، و بالتالى هى تأخذ نفس حكم الـ C# بلا زيادةٍ أو نقصان.

و نعود إلى اللغة الثالثة الشهيرة فى عالم الشمولية و هى الـC++، و التى حازت على مكانتها فى عالَم الشمولية بِلَى الذراع (إن كان لى أن أستخدم هذا التعبير)؛ فعلى الرغم من أن الـC++ لغةٌ تتميز بالشمولية و يمكن للمُبرمِج بها الإعتماد عليها فى كل المجالات البرمجية التى تخطر على باله (أو لا تخطر) لأى نظام تشغيلٍ أو مُعالِج تحت الشمس، إلا أن تلك الشمولية فى رأيى شموليةٌ خَدَّاعةٌ لا يُمكِن الاطمئنان إليها بحالٍ من الأحوال (على الأقل بالنسبة لى)؛ نظراً لتعقيد اللغة الذى يُضَيِّع أى ميزةٍ أخرى تمتلكها و على الأخص فى المجالات التى تتعقد فيها الخُوارِزْمات و الأفكار فلا ينقص الطين بلة تعقيد اللغة البرمجية نفسها.

و لكن لأننا فى هذا المقام نناقش الشمولية فقط فيمكننى أن أقول أن الـC++ تتربع على عرش الشمولية بجانب الـjava تماماً (و شتَّان ما بينهما فى التعقيد).

البساطة و الإستقرار:

مِن أكثر الأمور التى تُثير غيظى فى لغة الـC# أننى بعد أن قَضيتُ فترةً لا بأس بها أتعلمُها فيها و أتشرّبُ طريقة التفكير التى تتميز بها كسابقتها فى العمر الـjava ؛ أصبحتُ أُحِسُّ بأننى لا أعرف البرمجة بها على الإطلاق ! ؛ و يُراوِدنى الإحساس بهذا كلما رأيتُ مقالًا أو كُتَيِّباً يتحدث عن الإصدارات الجديدة التى تُصدِرها microsoft من اللغة كل فترة. فهذه الإصدارات أصبحت رُوَيْداً رُوَيْداً تَنفِخ فى كمية قواعد اللغة حتى انفجرت فى وجه المستخدمين الأبرياء (و لى الفخر فى أن أكون واحداً منهم).

و بعد أن كنتُ أتفاخر بأننى أفهم الـC# جيداً أَصبحتُ أَخافُ من التصريح بأننى أستخدمها فى البرمجة؛ حتى لا يقوم أحدهم بإحراجى حينما يَسألنى عن القاعدة الفلانية أو القاعدة العلانية التى ضُمَّت إلى اللغة فى الإصدارة ذات الرقم الفلانى فيبدو جهلى واضحاً له.
و الأمر أصبح مدعاةً للرثاء و تقليب الأكف حيرةً عند التفكير فى السبب الذى يقف وراء تلك "التطويرات" و النفخات الجديدة التى تُعطَى لِلّغة على فتراتٍ مُنتظمة، حتى أن نَحْو اللغة الآن لا يكفيه لشرحة بمُفرده (شرحاً احترافياً مُختصَراً) كتابٌ ضخم الحجم لا يقوى على إستيعابه المُبرمِج العادى إلا فى شهورٍ (أو علي الأقل أسابيع) طويلةٍ ينقطع فيها لدراسة اللغة تماماً.
و فى النهاية لا يكون أمامه إلا تَرْك جانبٍ كبيرٍ من اللغة لدراسته فى وقتٍ لاحق؛ نظراً لأنه لا يستطيع ترك باقى العلوم البرمجية و التفرغ لدراسة لغة البرمجة كل الوقت (و هى التى لا تمثل إلا جزءاً صغيراً من العمل البرمجى)، و الجدير بالذكر أن الوقت اللاحق الذى يُؤجِّل الدارس إليه دراسة ما بقي من اللغة لا يأتى أبداً، و ذلك عن تجربةٍ واقعية.

و هكذا لا يكون أمام المُبرمِج إلا خيارٌ من اثنين:
أولهما أن يتفرغ لتعلم اللغة فترةً كافيةً تُمكِّنه من التمكن الجيد منها، ثم مُتابعة التطورات و الإضافات الجديدة التى تَحدث و تُضاف للغة كل فترةٍ حتى يُصبح مُواكِباً لأحدث تقنياتها على الدوام.
و **ثانيهما** الإكتفاء بمعرفة الجزء الأكبر و الأهم من اللغة و التمكن منه جيداً، ثم الانطلاق إلى العلوم البرمجية الأخرى التى لابد من التمكن منها بجانب لغة البرمجة العملية.
أما الخيار الأول فيُنا سِب من بدأ مشواره فى تعلم البرمجة منذ أن كان صغيراً؛ و بالتالى كان أمامه الفترة الكافية للتعلم قبل مواجهة سوق العمل الذى يضغط الأوقات و الأعصاب إلى أقصى الحدود، و لا يُمكِن أن يتناسب مع ظروف شخص مثل كاتب هذه الأسطر على سبيل المثال؛ فالعمل يدفع من هم مثله للقراءة و التعلم و الإحتراف فى مجالاتٍ أخرى غير لغة البرمجة المعنية، بل وربما يَدفعه سوق العمل على إتقان لغةٍ أخرى غير اللغة التى يَهتم بها و يجد أنها تُناسِب نَمَط تفكيره.

و أنا مثالٌ جيدٌ على ذلك: فقد كنتُ أستخدم فى البرمجة لغة الـC++ التى أمقتُها بعنف، و كنتُ مُطَالَبَاً باحترافها و التمكن منها حتى أستطيع العمل بها بحرفية فى مجال مُعالَجة الصور الرقمية، و رغم أنني أحب الـjava و الـC# إلا أننى لا أستطيع مُجاراة آخر التحديثات و التغييرات التى تتم لهما؛ لأننى لا أستطيع أن أقضى وقتى كله فى تعلم الـC++ من ناحيةٍ و تعلم الجديد فى الـC# من ناحيةٍ أخرى، في حين أن هناك علوماً أخرى يجب أن أُحَصِّلها حتى أكون مُبرمِجاً بحق (مثل هندسة البرمجيات، و الخُوارِزْمات، و غيرهن) و التى لم أتعلمها من قبل لأننى احترفت مجال البرمجة و الحوسبة عموماً منذ فترةٍ لا تزيد على الخمس سنوات.

و هناك من يُناسِبه خيار المُتابَعة المُستمرة للُّغة (غير الذي بدأ فى التعلم منذ الصغر) و هو: مَن بدأ مع اللغة منذ بداياتها الأولى و أول إصداراتها، و بالتالى كانت لديه الفرصة الكاملة للمُتابعة الدائمة كأفضل ما يكون، و هؤلاء ليسوا بالقليلين بل هم فى كل مكانٍ حولنا، و يُمكِن أن نراهم كثيراً حينما يأتى الأمر على لغةٍ مثل الـC# التى أصبحت البرمجة بها موضةً من موضات المُبرمِجين في السنوات الماضية، و بالتالى انهمك الجميع فى دراستها و هى بعد فى المهد، و بالمُتابَعة تعلموا الجديد أولاً بأول.

أما الخيار الثانى القائل بالتمكن من الجزء الأكبر و الأهم من اللغة فقط لكى يتبقى الوقت اللازم لتعلم باقى العلوم البرمجية فهو أمرٌ فيه جدالٌ يَطول؛ و ذلك لأنه و إن كان مع أصحابه الحق فى القول بأهمية تعلم باقى العلوم البرمجية غير لغة البرمجة ذاتها فإنهم لا ينتبهون إلى نقطةٍ أخرى فى غاية الأهمية: و هى أنهم بتطبيقهم خيار التجزيئ للُّغة لن يتمكنوا من فهم الأكواد البرمجية التى يكتبها غيرهم ممن يُبرمِجون بنفس اللغة البرمجية و لكنهم يستخدمون مُكوّناتٍ لم يتعلموها هم.

ولن يكون مثل هذا الموقف قليل الحدوث بل سيحدث كثيراً جداً كلما زادت كمية الأكواد التى يضطلع عليها المُبرمِجون، و هى كثيرة العدد جداً فى هذه الأيام بفضل عالم المصادر المفتوحة و الحاجة المضطردة للإعتماد عليها حتي لو علي سبيل التعلم.

و هذا يعنى أن أمثال هؤلاء المُبرمِجين سيُجبَرون إن عاجلًا أو آجلًا على التعرُّف على مُعظَم المُكوّنات التى تركوها خلفهم فى اللغة إن لم تكن كلها، و بالتالى يكون من الأفضل لهم أن يفعلوا ذلك من البداية فى ذروة تعلم اللغة ما دام الأمر لا مفر منه فى النهاية، و من ثم يكون ما نخافه من الإهمال لباقى عناصر العلم البرمجى أو حتى عدم التنبه الكامل لها حتى الانتهاء من تعلم لغة البرمجة (التى هى الأداة التى يستخدمها المبرمج فى عمله ليس إلا).

و هكذا نرى بأم أعيننا الوقت الكبير يَضِيع فى تعلم ما كان يجب أن يكون هو الأمر الأبسط فى العملية التعليمية كلها، بينما الأمور الأهم تُترَك لما بعد، و بعد كل ذلك تأتى مرحلة التطبيق على المجالات التى سوف تُستخدَم فيها كل تلك الخبرات البرمجية مثل برمجة الشبكات أو برمجة تطبيقات معالجة الصور الرقمية. و ليس لنا بعد كل ذلك الوقت المُهدَر التعجب من كره الكثيرين من التعمق فى

دراسات البرمجيات النظرية مثل هندسة البرمجيات و الخوارزمات و رؤيتهم لها على أنها غير ذات جدوى، بينما ينظرون إلى تعلم لغة البرمجة على أنه هو كل شئٍ فى عالم البرمجة ما دام يأخذ وقتاً كبيراً و يأتى دائماً فى المقدمة لكل شئ، و يُركِّز عليه الجميع أغلب اهتمامهم.

و هكذا نرى أيضاً كل عام أكواماً من الكتب التعليمية التى لم تعد لها أى فائدةٍ رغم الجهد الذى بذله مؤلفوها فيها؛ لأنها أصبحت تتحدث عن ماضى اللغة لا عن حاضرها، و رغم المال الذى دفعه فيها مُشتروها الراغبون فى تعلم اللغة، و بالتدريج سيكون على المُتعلِّم التفرقة بين الكتب القديمة و الحديثة بمُنتهَى الحرص حتى لا يقع فى معلوماتٍ مغلوطةٍ عن اللغة.
و مثل هذا الموقف نراه فى الـC# التى كنا منذ فترةٍ ليست بالبعيدة نقول عنها أنها لغةٌ ذات أنواع ثابتة أو (statically typed) و من ثم فاجأتنا الإصدارات الجديدة أنها صارت أيضاً ذات أنواعٍ متغيرة أو (dynamically typed).

***

و هذا الحال بالطبع لا يُرضِينى ولا أقتنع بجدواه على الإطلاق، بل أقول أن كل هذا عبثٌ سخيفٌ ليس له أن يستمر و لا يحق لأحدٍ أن يفرضه على الآخرين بأى داعٍ من الدواعى، فمنتجوا لغات البرمجة الشهيرة الذين يزيدون فى قواعدها يوماً بعد يوم مثل microsoft و لغتها الـC# يجب عليهم أن يتوقفوا حالاً عن هذا و إلا كان لزاماً على معاشر المبرمجين مقاطعة أمثال تلك اللغات تماماً فيصير أمثال أولئك المُنتِجين مثالاً لكل من يتحدى إرادة مجتمع المبرمجين و يجبرهم على اللهاث وراءه فى عمليةٍ تعليميةٍ تعذيبية الطابع لا نهاية لها.

الأسباب التى تجعل مُنتِجي لغات البرمجة يفعلون هذا:

إذا ما فَكَّرنا فى هذه المسألة بعمقٍ لوجدنا أن الأسباب التى تَجعل مُنتجي لغات البرمجة يزيدون من قواعد لغاتهم أو "يطورونها" ربما تَختلف حسب المنهجية التى يَنظر بها ذلك المُنتِج إلى المُجتَمع البرمجى نفسه، و لنا فى لغة الـC# خير مثالٍ على ذلك، فمُنتِج لغة الـC# هو microsoft التى لا تسعى (مثلها مثل كل الشركات التجارية) إلا وراء الكسب المادى. و للحصول على المكسب المادى من وراء لغة الـC# من المهم أن يكون لها أكبر عددٍ من المُستخدِمين من المُبرمِجين مُختلِفى الطِباع و الأهواء، لذا نَرى أن الـC# تَحوى داخلها من الصفات ما يَجعلها تَجميعةً من مُختَلف أنواع التفكير البرمجية، و يَزداد هذا الأمر وضوحاً يوماً بعد يوم بالزيادات التى تُوضَع فيها بدعوى التطوير و التحديث.
و قد ضربنا لذلك مثالًا بالـstatic typing و الـdynamic typing، و نَضرب له مثالاً آخر هو البرمجة الآمنة و البرمجة غير الآمنة أو الـsafe programming و الـunsafe programming، و التى تُمَكِّن مُبرمِجى الـC# من استخدام المُؤشِّرات التى تَعوَّدوا عليها فى الـC++ (بالطبع مع وجود بعض الإختلافات الضرورية). وهو الأمر الذى قد يُسعِد بعضاً من

مُبرمِجى الـ++C و يُقنِعهم بالفعل بالإنتقال إلى البرمجة بالـC#، و لكنه فى نفس الوقت يَجعل الـC# بالنسبة لمن هم مثلى خدعة كبري؛ لأننا في الأ صل لم ننتقل للC# إلا لأنها لغةٌ مُتطورةٌ و لا تحتوي علي المُكوّنات التي يكرهونها في عالم الـ++C و علي رأ سها المُؤشِّرات، فكيف بعد هذا نَجِد أن مايكرو سوفت "بِتِبْلِف"[7] !

عدم الخلط بين الاستقرار و الجمود:

على أن كل الكلام السابق الذى نَستنتجُ فيه ضرورة الإستقرار كعامِلٍ أساسي من الضروري تَوافرُه فى لغات البرمجة القوية لا يَجبُ أن يَدفعنا إلى الظن بأن الجُمود سيكون هو مَصير اللغة التى تُطَبِقه على المدى البعيد و تُصِر على هذا التطبيق؛ فالواقع أن الأمر غير ذلك نهائياً: لأن الإستقرار ليس معناه الثبات التام للّغة عند نَحْوٍ مُعيّنٍ منذ البداية كما قد يكون قد فهم الكثيرون من سابِق الأقوال، بل المقصود بالإستقرار هو: عدم التغيير إلا عند الضرورة المُلِحّة التى لا يُمكِننا الإستجابة لها إلا بالتغيير **الطفيف** في اللغة.

و أقول التغيير **الطفيف** لأن التغييرات الكبيرة يجب أن تَدفعنا دفعاً إلى التفكير فى تصميم لغة برمجةٍ أخرى مُنفصِلةٍ تماماً عن اللغة التى صارت من طِرازٍ أقدم، و تَحتوى داخلها المُكوّنات الجديدة التى نَرغب فى ضمها إلى اللغة الأقدم.

و هذا هو الإستقرار الذى أرغب فى الوصول إليه فى لغات البرمجة القوية، و أرى أنه من أول الصفات التى يجب أن تتوفر فى أى لغة؛ فلغة البرمجة القوية جيدة التصميم هى فى حقيقة الأمر لغة البرمجة التى لا تحتاج إلى أى تعديلاتٍ على نَحْوِها إلا على فتراتٍ مُتباعِدة، و كما قلنا قبلاً فستكون تلك التغييرات قليلة العدد حتى لا يزداد نحو اللغة ضخامة، و الأهم أنها ستكون طفيفةً لا تَمَس هيكل اللغة أو نُموذجها البنائى بأى حال.

أما لغة البرمجة التى ستحتاج إلى تعديلاتٍ كثيرةٍ أو جوهريةٍ على فتراتٍ مُتقاربةٍ فهى لغةٌ سيئة التصميم من البداية، و كان يجب ألا يتم التعجل فى تصميمها، أو كان يجب ألا يتم الإعلان عنها من الأ صل حتي تنضج، و لو طُبِق هذا المبدأ البسيط على لغات البرمجة من البداية لأراحنا من كثيرٍ من أوجاع الرأس.

و فى النهاية فإننى ألخص موقفى فى تلك النقطة الجوهرية من مَبْحَث تصميم لغات البرمجة فى البنود التالية:

- يجب أن تكون اللغة مُصمّمةً على "نارٍ هادئةٍ" حتى لا نحتاج إلى التغيير الكبير فيها فيما بعد.

---

7 كلمة عامية مصرية معناها (تَخْدَع) أو (تَغُشْ)، مع إعطاء إيحاءٍ قويٍ أن الفاعل يعتبر الآخر أحمقاً مُغَفّلًا.

- يجب أن تَضُم اللغةُ المُكوّنات التى لا يُمكِن الإستغناء عنها و لا يُمكِن أن تحل محلها مُكوّناتٌ أخرى تم ضمها بالفعل للّغة، أى أن الأولوية فى الضم ستكون للمُكوّنات التى لا غِني عنها و التى تُغنِى عن غيرها بينما يتم رفض ذلك الغير.

- إذا ما كانت هناك تطوراتٌ فى البرمجة تجعل مُكوّناً جديداً جديراً بالضم إلى اللغة فيجب أن يتم حسم مسألة كَوْن هذا المُكوّن الجديد لا يَهدم مُكوّناتٍ أخرى أم لا، فإن كان لا يفعل و كان سهل التأقلم مع المُكوّنات الموجودة بالفعل: فيُمكِن ضمه إلى اللغة بعد فترة استقرارٍ جيدةٍ لا تقل فى رأيى عن عامٍ كاملٍ عن التغيير الضرورى السابق مُباشرة.

- على مدى حياة اللغة يجب أن يكون هناك حدٌ يتم بعده رفض ضم أى مُكوّنٍ للّغة مهما كانت أهميته، و هو ما يمكن أن نُسمّيه "حد التُّخْمَة"، و أُقدِّر أنا هذا الحد بحجم الكتاب الذى يشرح نَحْو اللغة على نحوٍ مُحترفٍ مُختصَر، فلو زاد حجم الكتاب على الخمسين صفحةً من ورق الـ(A4) المكتوب بخطٍ وَسَطٍ لا صغيرٍ و لا كبير (حجم شرح قواعد إبداع) فيُمكِننى أن أعتبر أن هذه اللغة قد بلغت حد التُخْمة، و يجب التقليل من كل ذلك الكم من القواعد التى لا داعى لها.
  و رغم أن الأمر يخضع للتقدير الشخصى إلى حدٍ كبيرٍ إلا أنه بالحس الشخصى يُمكِن التوصل إلى نتائج شديدة الشبه عند النظر إلى لغات البرمجة المُختلِفة من عينى أشخاص مختلفين إستناداً إلى المعيار السابق للتُخمة، أى أنه رغم كَوْن المِعيار غير فائق التحديد إلا أنه عملياً سيكون ذا قوة توحيدٍ قياسيةٍ كبيرةٍ بين المُبرمِجين و الناقدين للغات البرمجة.

- عندما تبلغ اللغة حد التُخْمة و يكون هناك مُكوّناتٌ **لابد** من وجودها فى لغة البرمجة الجيدة لتُلائِم الأفكار الجديدة فى العمل البرمجي، أو عندما نحتاج إلى تغيير أحد الأفكار **الرئيسة** فى لغة البرمجة، أو حتى تكون هناك مُكوّناتٌ عَفَى عليها الزمن و **يجب** التخلص منها، فيجب علينا أن نُفكر فى إنتاج لغة برمجةٍ جديدةٍ تستلهم الجيّد الذى فى اللغة القديمة و تضع الجديد المرغوب فى ضمّه محل القديم المرغوب فى تركه.
  و هذا يجعل تصميم اللغات أسهل، و تطورَها أكثرَ قابليةً للفهم، كما أنه يُعطِى للراغبين فى البرمجة باللغة على نفس الشكل القديم لها الفرصة لفعل ذلك، و لا يُجبِرهم على التغيير للشكل الجديد ما دام الشكل الجديد قد صِيغ على شكل لغةٍ برمجيةٍ أخرى مختلفة، لهم كل الحق فى التقرير لأنفسهم إن كانوا سيتعلمونها و يستخدمونها أم لا.

الأَمْن:

ليس المقصود بالأمن هنا الحماية ضد البرمجيات الضارة باختلاف أنواعها من فيروساتٍ و برمجيات تجسسٍ و أحصنة طروادة، بل المقصود به: وجودُ رقابةٍ من مُصمِّم اللغة على المُكوِّنات التى تُضَم لها، بحيث يَضمن كونها غير قابلةٍ لأن تكون مصدراً للأخطاء أو القلاقل فى البرامج التى تُستخدَم فيها، و كذا ضمُّ المُكوِّنات التى تُساعِد المُبرمِج على التغلب على الأخطاء التى تَظهر أثناء عمل البرنامج مثل مُعالَجة الأغلاط.

أى أن الناحية الأمنية فى اللغة تتكون على الأقل من عنصرى: تَلافِى المُكوِّنات السيئة و ضمّ المُكوِّنات المُفيدة، و على هذا فيُمكِننا أن نَرى أن عنصر الأمن يَصب فى ناحية عنصر البساطة و الإستقرار؛ حيث يَضع معاييراً تَحُد من الأعداد التى يتم قبول ضَمّها إلى اللغة من المُكوِّنات البرمجية، فيُحافِظ على كون اللغة فى الحد الأقصى المقبول لها من الحجم على الأقل، كما يَضمن أن القواعد التى ضُمَّت إليها هى بالفعل الأجدر بالضم.

و يُمكِننا أن نَضع الأمن اللغوى بتوسعٍ و شمولٍ فى ثلاثة عناصرٍ هى:

- وُجود مُكوِّناتٍ تُسَهِّل العمليات كثيرة الإستخدام، و التى قد يُخطِئ فيها المُستخدِم إذا ما كانت تتطلب كثيراً من الجهد نظراً لكثرة تكرارها، و بالتالى فإن تسهيل العملية على المُبرمِج بتوفير المُكوِّنات الجاهزة فى لغة البرمجة التى تقوم بالعملية كاملةً بالنيابة عنه لا يخدم فقط الناحية الإنتاجية فى العملية البرمجية، بل كذلك يخدم بشكلٍ أساسيٍ عملية تفادى الأخطاء البرمجية.

- إستبعاد المُكوِّنات أو التعبيرات البرمجية التى تُسبِّب البلبلة و الارتباك و من ثم الأخطاء البرمجية بكل أنواعها من أخطاءٍ نَحْوية و أخطاء زمن تشغيلٍ و أغلاطٍ منطقية، و مهما جادَل المُجادِلون فى فائدة مثل تلك المكونات أو التعبيرات فيجب أن يُوقِنوا أنه يُمكِن الإستغناء عنها؛ لأنها نِتاج فِكرٍ بشرى وجدها حلاً لما واجهه من مشاكل، و ليست تنزيلاً من السماء يجب قبوله كما هو، و يُمكِن بالقليل من التفكير (أو حتى بالكثير منه) إبتكار مُكوِّناتٍ أو أساليبٍ جديدةٍ تُغْنِى عن تلك المُكوِّنات و التعبيرات المُربِكة.

- وُجود المُكوِّنات التى مهمتها الأساسية مُعالَجة الأخطاء في زمن التنفيذ (و التي يتكون منها أسلوب مُعالَجة الأغلاط) فى الـjava و الـC# و كثيرٍ من اللغات الأخرى، و هذه المُكوِّنات تجعل عملية البرمجة متعةً حقيقيةً لأنها تُقرِّب الأمر من التفكير العالى للبشر إلى

أقصى الحدود. فبدلًا من البحث اليدوى عن كل الأغلاط المُتوَقَّع حدوثها و كتابة الأحداث اللازمة لمُعالَجتها: أصبح لدينا تلك الصياغة التى يُمكِننا بها ببساطةٍ تحديد الإجراء المُناسِب لكل نوعٍ من أنواع الأغلاط.

الأُلفة:

في مجلة (مجتَمع لينوكس العربي) و بالتحديد في العدد رقم 7 الصادر في رمضان 1430هـ (الموافق أغسطس 2009م) كانت هناك قصةٌ قصيرةٌ من ضمن سلسلة قصصٍ قصيرةٍ تتحدث عن مُغامَرات مُبرمِج مُخضرَم في لغة بيرل perl. و اسم تلك السلسلة هو (من مُغامَرات المُحقِّق وميرت فونلي)، و كانت قصة ذلك العدد تُسمّى (سطرٌ بلغة بيرل) حيث فيها تَعرَّض واحدٌ من المُبرمِجين المُبتدئِين لمُشكلةٍ ما، فطلبَ من المُحقِّق (وميرت فونلي) مُساعَدته على حلها، و هنا فَكَّر المُبرمِج الخبير بعض الشئ و من ثم خرج ببرنامجٍ بلغة perl يحل المسألة كلها، و الأهم أنه من سطرٍ واحد !

و الحقيقة أنني منذ أن قرأتُ تلك القصة (و حتى الآن) و أنا أضعها في رأسي مثالًا للإدعاء و السخافة؛ لأن القصة تُحاوِل القول أن لغة perl قويةٌ و ذات إمكانياتٍ شديدة القوة لدرجة أنه يُمكِنها أن تحل مشكلةً صعبةً بسطرٍ برمجيٍ واحد. و لو كان الأمر هكذا لما كان هناك ما يَضر في الأمر و لكنتُ واحداً من أكثر الناس إسراعاً للبرمجة بـperl. و لكن الأمر كان على خِلاف ذلك تماماً، فنظرةٌ واحدةٌ إلى البرنامج ذي السطر الواحد و هو:

```
perl -MDigest::MD5=md5 -
0we'@a=@ARGV;@h{map{md5($ )}<>}=@a;@b=values%h;print"@b\n"' *
```

هذه النظرة الواحدة تكفي لنرى أننا قد خُدِعنا و دُلِّس علينا حينما زَعم المؤلف أن هذا برنامج حاسوب؛ بل كان (و اعذروني في التعبير) نبش الدجاج أو طلاسم السحرة، فالبرنامج السابق لا يَحوي من مُوا صفات البرامج المُهَنْدَسة جيداً شيئاً و لا حتى على أقل القليل منها، فهو:

- صعب القراءة و الفهم إلى درجةٍ كبيرةٍ على الخبير بلغة perl؛ لتَرَاكُب التعبيرات بما جعله بالفعل شبيهاً بأحجية السحرة و طلاسمهم. فما بالنا بالمُبرمِجين البُسطاء الذين هم عامة المُبرمِجين ؟!، و الدليل على صعوبة فهم البرنامج أن شرحه الموجود في القصة كان أكبر من البرنامج بمراحل، حيث وصل إلى ما يُقارِب الصفحتين الكبيرتين بينما لم يأخذ البرنامج نفسه إلا سطراً واحداً !

- صعب الصيانة و التطوير؛ لأن تعقيده سيأخذ من المُتابِع له وقتاً كبيراً لفهمه أولًا، ثم سيأخذ الوقت الأطول للتغيير فيه عند الرغبة في التطوير، و لأنه مُعقَّدٌ فإن التغيير فيه مغامرةٌ محفوفةٌ بالمخاطر لن أَلِجَ (عن نفسي) مثلها أبداً إلا مُرغَماً.

و لأنه من أول مُوا صفات البرامج الجيدة هو أن تكون واضحةً للقارئ؛ بحيث تكون فى النهاية سهلة الفهم و سهلة التطوير قدر الإمكان: فإن البرنامج السابق يرسب فى معايير البرمجيات الجيدة بمُنتهَى الجدارة.

و لكننا نظلم البرنامج إذا فَكَّرنا أن المشكلة فيه هو فقط؛ لأن المُشكِلة الأساسية فى اللغة التى كُتِب بها البرنامج نفسه و هى لغة perl، فهذه اللغة من أكثر اللغات التى تحتوى على رموزٍ و علاماتٍ تُبعِدها عن الشكل القريب من الفهم البشرى و تُقرّبها من الشكل الطلسمى.

و كثرة العلامات التى تعمل عمل البديل للنصِ البرمجى أمرٌ لا يُساعِد إلا على إنتاج برامج خُرافِيةٍ لا تُصدَّق فى تعقيدها و شكلها الأسطوريين، تماماً كالبرنامج السابق الذى نرى الأمرين قد تحققا فيه.

و صلب المُشكِلة هو أن العلامات و الرموز التى لا عمل لها فى الحياة العادية للإنسان قد أخذَت حجماً مُبالَغاً فيه من صِياغات اللغة و قواعدها، على حساب الكلمات النَّصِّية العادية التى هى أقرب للفهم و الإدراك البشريين، و السبب الوحيد الذى قد يدفع إلى هذا (عند مُصمِّمي ذلك النوع من اللغات) هو كَوْن تلك الرموز و العلامات أسرع فى الكتابة و القراءة، و بالتالى فإنها تُؤدِّى إلى كِبَر الإنتاجية و هو هدفٌ يسعى إليه الجميع.

لكن المُشكِلة أن من يَقتنِع بهذا الرأى يَرتكِب ثلاثة أخطاءٍ كبيرةٍ هى:

- إعتَقَد أن البرامج صغيرة الحجم هى بالضرورة سهلة التَّقبُّل و الإستيعاب، و قد قُلنا أن هذا خطأٌ لأن الإسراف المَرَضِى فى العلامات الغريبة على العقل البشرى فى حياته اليومية سُيحيل البرامج (بالنسبة له) إلى تعاويذٍ سِحريةٍ و طلاسم، فيَقضِى أمامها أغلب الوقت لفكّها و فهم مُرا د الساحر (أقصد المُبرمِج) منها.

- إعتَقَد أن الإنتاجية تتأثر أول ما تتأثر بسرعة الكتابة ! و هو تصورٌ طفولىٌ لا يجب أن يَخطر إلا ببال المُبتدئين من المُبرمِجين، الذين لا يَرون من البرمجة إلا ما يَرونه فى مسائل المنهج الدراسى الذى يدرِسونه فى الجامعة، أو مسائل الإمتحانات بأفكارها البسيطة المُكرَّرة لا أكثر و لا أقل. فأصبَحَت كل البرمجة بالنسبة لهم معرفة المطلوب منهم، و من ثم الجلوس أمام الحاسوب و البدء فى الكتابة بسرعةٍ ليُمكِنهم الحل و المُراجَعة قبل إنتهاء الوقت المحدد.

- أهمَل وُجود الأدوات البرمجية المُساعِدة الحالية التى تجعل بإمكانياتها القوية اللغاتَ التى تُسرِف فى إستخدام الرموز البسيطة و اللغاتَ التى تُسرِف فى إستخدام الكلمات الطويلة على قَدَم المُساوَاة، و عن تجربةٍ مع لغة الـ#C و بيئة التطوير المتكاملة الـVisual studio .NET أقول أن أمر الكتابة السريعة كان بالنسبة لى أمراً منطقياً بَدَهِياً لما تُوفّره لى بيئة الـVisual studio .NET من إمكانياتٍ مُساعِدة، و اقتراحاتٍ فى كل خطوةٍ

أخطوها، و هو ما كان يَجعلنى أُركِّز جل تفكيرى (بل كله) على الأفكار و الخُوارِزْمات التى سأستخدمها فى البرنامج ليُؤدِّى مهمته بكفاءةٍ و قوة.
بل إن الأمر فاق هذا إلى المُساعَدة فى إنتاج الكود للأجزاء التى تَحتاج إلى وقتٍ كبيرٍ من المُبرمِج لعملها، بينما يُمكن للحاسوب مُساعَدة المُبرمج على القيام بها بكفاءةٍ و سرعةٍ شديدين، مثل مُصَمِّمَات الواجهات الرسومية GUI designers التى توشك بيئات البرمجة المُتكامِلة جميعها على الإتفاق على ضمها فيها؛ لما تُزيحه من عبء تكوين شكل النوافذ باستخدام الكود مُباشرة، بينما باستخدام مُصَمِّمَات النوافذ فإن كل ما علي المُبرمِج هو أن يَسحب المُكوِّن الذى يُريده و يَضعه فى المكان الذى يشاء ببساطةٍ و سلاسةٍ شديدين.

و إنى لألاحظ أن إهمال تلك الإمكانيات يُوشك أن يكون آفةً فى جَمْعٍ كبيرٍ من المُبرمِجين المُخضرَمين، بما يُوحِى بأنهم يَتصورون أن استخدام مثل تلك الإمكانياتً سيُقلِّل من كفاءتهم، لأنها أدواتٌ للمبرمجين المبتدئين الذين لا يُمكِنهم الإعتماد على أنفسهم كل الإعتماد فى بناء و كتابة أكوادهم !. و هى النظرة التى أضمُّها بحماسٍ إلى قائمة الأمور التى تُثير جنونى فى عالم البرمجة؛ فهذه الأدوات معلومٌ من النظر و الخبرة بالضرورة أنها ذات أثرٍ بالغ القوة فى العمل و الإنتاج البرمجي فى المشاريع الكبيرة، و تُؤدِّى إلى طفراتٍ إنتاجيةٍ كبيرة، مما يَعنى أنه من العقل السليم استخدامها و الإستفادة منها بأقصى حدٍ مُمكِن. و بالتالى يكون التصور السابق أسخف من أن يَخطر ببال أحدٍ من الأصل فما بالك باعتناقه و تطبيقه فى مجال تصميم لغات البرمجة ؟! و من ثم فَرْض نتائجه علي الجيل الجديد من المُبرمِجين !

عدم الخلط بين الألفة و الإطناب المرضى:

مَدْحُنا السابق فى كَوْن اللغة البرمجية قريبةً من الكلام البشرى لا يجب أن يدفعنا إلى الظن أن القُرْب الكبير من الطريقة البشرية العادية فى الحديث أمر مُحبَّبٌ فى البرمجة؛ بل الأمر على العكس من ذلك من حيث أن ذلك القُرْب الكبير يُحوِّل البرنامج إلى عجينةٍ من الكلمات التى لا يُمكِننا أن نُحدد من بينها ما نُريد من معلوماتٍ إلا بعد عناءٍ و جهد، فالمسألة تَحتاج إلى تَوسطٍ فى الإختيار عند تصميم اللغة البرمجية بين اختصار و صِغَر حجم الرموز و العلامات و بين وُضوح و مَفهومية الكلمات النصية.

و من الأمور التى تَضبط هذه المسألة و يُمكِن الإحتكام إليها القواعدُ التالية:

- **قاعدة ضمّ المألوف من العلامات**، فلا يُضَمّ رمزٌ أو تُضَمّ علامةٌ إلا إذا كان/ت مما أَلِفه المُستخدِمون فى تعاملاتهم مع النصوص العادية، مثل الفاصلة ، و النقطة . و القوسين ()، والتقليل قدر الاستطاعة من الرموز و العلامات التى يَقِل التعامل بها أو يَنعدم فى الحياة العادية مثل ~ و *

- إستخدام الرموز و العلامات المألوفة (لو احتجنا لضَمّهما) فى نفس الوظيفة التى كانت تُستخدَم فيها فى اللغة العادية، مثل استخدام النقطة . فى نهاية الأوامر البرمجية و استخدام النقطتين القائمتين : عند بدء تعبيرٍ أو تركيبٍ من الجُمَل.

- إستخدام الكلمات بدلًا من الرموز فى الأماكن المُناسِبة لهذه الرموز فى الأحيان التى تَكون فيها للكلمات القدرة على القيام بوظائفٍ زائدةٍ لا يُمكِن للعلامات القيام بها، فمثلًا فى إبداع استخدمتُ الكلمات المفتاحية و أو ليس للإستخدامات المنطقية فى اللغة، على الرغم من أنه كان بالإمكان استخدام العلامات المشهورة فى لغات البرمجة الإنقليزية مثل && || ! فى ذلك و كانت ستُؤدّى دور الكلمات المحجوزة، إلا أن هناك وظيفةً لا يُمكِن لهذه العلامات أداؤها، و هى تحويل النص البرمجى إلى لغةٍ قريبةٍ من اللغة العادية مما يَزيد من مقروئية الكود بما لا يُوصَف بالنسبة للمُبتدِئين و الأطفال الصغار، الذين يُمكِن أن تُستخدَم اللغة فى تعليمهم البرمجة بإذن الله تعالى.

المُكوِّنات التى رُفِض ضَمُّها إلى اللغة

أَرفقتُ هذا الجزء بالرسالة لرغبتي في تقديمه كتطبيقٍ عملي و شرحٍ واقعي لما تُلزِم به المبادئ التصميمية (التى سبق و أن أبديتُ اقتناعى بها فى الجزء السابق) كلّاً من يَسِير بمُقتَضاها، و الأثر العميق الذى ستتركه على عقله عند نقد المُكوِّنات المعروفة و المتوافرة حالياً فى اللغات البرمجية الأخرى، و التى لا شك فى كونها بيئةً خصبة للتعلم و النقد بالنسبة له.

لذا وضعتُ هنا أهم المُكوِّنات البرمجية التى رَفضْتُ ضمّها للغة إبداع تأثراً كما قلتُ قبلًا بقناعاتى التصميمية السالفة الشرح، و من الأمور التى سيلاحظها كل من له علاقة احتكاكٍ بعائلة لغات الـC أن مُعظَم هذه المُكوِّنات تُوجَد فى تلك العائلة، أو على الأقل فى مُعظَم أفرادها إن نجا منها واحدٌ عند مُناقَشة مُكوِّنٍ ما مِن ضِمْن ما ذُكِر هنا.

و السبب الرئيس هو أننى قد تلقيتُ مُعظم تعليمى الجامعى (و ليس التعليم الذاتي) على تلك العائلة، و لم يَشذ عن تلك القاعدة إلا لغة الـvisual basic التى دَرَستُها فى الفرقة الإعدادية بشكلٍ طفيفٍ للغاية لم يُؤثِّر فِيّ أبداً كما أثَّرت فِيّ لغات تلك العائلة، و كذلك لغة الـfortran التى كرهتُها بعمقٍ حتى اليوم، و لكنى لم أر فيها مُكوِّناتٍ تُرفَض من الأصل؛ و ذلك لكونها من الأجيال الغاية فى القِدَم برمجياً و بالتالى ليس هناك ما يُرفَض فيها لأنها تُمثِّل كلها كماً مرفوضاً (كما سيتضح لأى مقارنٍ بين إبداع و الـfortran)، على الرغم من أننى قد تأثرتُ بها تأثراً بسيطاً للغاية فى مسألةٍ تخص التعامل مع عناصر الجداول فى إبداع.

كما أن عائلة الـC تضم مجموعةً من اللغات الأشهر فى العالم البرمجى كله على اختلاف أذواقه و مشاربه، بدايةً من العالم السفلى بكل كهنته اللذين تُمثِّلهم لغتا الـC و الـC++، وُصولاً إلى العالم الأعلى بكل طياريه و تُمثِّلهم الـjava و الـC#. و هو الطيف الذى يجعلها تَضم مُعظم مُبرمِجى العالم (و لا عجب؛ فهن أربع لغاتٍ تَحتل كلٌّ منهن مكانةً فائقة القوة بين لغات البرمجة الأقوى).

و أُنوِّه هنا أن القارئ سيجد كلاماً مُكرَّراً فى أثناء النقاش هو للتنبيه على الأهمية القصوى لما يُنَاقَش أولًا، ثم لكى يكون بإمكان القارئ الكريم قراءة هذا الجزء مُفرَداً بدون أن يفقد الكثير من النقاط المحورية مما سلف ذِكْره، بل يكون بإمكانه اشتفافها من بين السطور على الأقل.

و المُكوِّنات المذكورة فى هذا الجزء هى على الترتيب:

1- الواجِهات interfaces،
2- الهياكِل structs،
3- جُمْلة goto،
4- المؤشِّرات pointers،
5- الإجراءات procedures،
6- الوسائط delegates،
7- المهمات tasks،
8- المُغَلِّفات Properties،
9- المُفَهْرِسات indexers،
10- نُموذَج الـPython للأصناف و الإجراءات،
11- الدامِجات unions.

الواجِهات interfaces:

نبذةٌ بسيطة:

الواجهات أو الـinterfaces فى الـjava و الـ#C هى مُكوِّنٌ شبيهٌ بالصِنْف العادى فيما عدا أنه قد لا يحتوى على بِناءٍ implementation لأيٍ من (أو كل) الإجراءات المُعرَّفة فيه، أى أنه فى الواجهة يُمكِن للمُبرمِج أن يُعلِن عن المُتغيِّرات التى يُريدها و يضع رؤوس الإجراءات التى يُريدها كما يفعل مبُرمِج الـ++C/C فى ملفات الترويسة header files لا البناء الخاص بالإجراء، و بعد ذلك يُمكِن للمُبرمِج استخدام تلك الواجهة فى الوظائف التى وُضِعت لها.

أسباب وجودها و الرد عليها:

- **أنها تعطى الهياكل إمكانية الوراثة:**

بدايةً فإن وجود الهياكل نفسها يُعَد عيباً فى اللغة فى اعتبارى كما سأُوَضِّح فيما يلى، لذا ففى إبداع ليس لدينا (الهياكل) من الأصل، كما أن هذا الكلام خِداعٌ مَحْضٌ لأنها وراثةٌ مُزيَّفة؛ فالوراثة ما هى إلا إمكانية أن تكتب الكود مَرَّةً واحدةً ثم بجملةٍ بسيطةٍ تقوم بإعادة استخدام ذلك الكود مَرَّة ثانيةً و ثالثةً و غيرهن بمنتهى الحرية مع الإستفادة بالتطوير المُستقبلى فى الكود الأصل. أما ما يحدث مع الواجهات فهو أن المُبرمِج يَضع تعريفاتٍ للمُكوِّنات الداخلية للصِنْف فى واجِهةٍ مُعيَّنة، ثم يقوم فيما بعد عند "وراثة" تلك الواجهة ببناء كل تلك المُكوِّنات من البداية بدون أى توفيرٍ للوقت أو الجهد !.

فأين هى تلك الوراثة المزعومة ؟!.

إذاً يتَّضِح لنا أن هذه المِيزة ما هى إلا فقاعةٌ فارِغةٌ تُقال و لا حقيقة لها، و السبب الوحيد لقولها هو دعم السبب الحقيقى لاستخدام الواجهات، و جعل الأسباب كثيرةً تملأ ناظرى القارئ لتُقنِعه بيسرٍ بأهميتها، فى حين أنه ليس هناك سببٌ حقيقيٌ يَستحق النظر إلا واحداً أو اثنين على الأكثر.

- **تُعطِى الأصناف خاصية الوراثة المتعددة التى لا تُوفِّرها لها الأصناف المُجَرَّدة:**

يلحق هذا السبب بسابقه من حيث أنه سببٌ مُزيَّف؛ فكما قُلنا فى السطور الماضية فإن تلك الوراثة (إن كان لنا أن نُسمِّيها وراثة) زائفةٌ لا حقيقة لها.

ثم لماذا لم يتم دعم الوراثة المُتعددة الحقيقية ما دامت لها فوائدٌ هامَّةٌ إلي هذه الدرجة ؟!

لقد تم دعم الوراثة المُتعددة في إبداع بالفعل، و بمُقارنته بدعمٍ الواجِهات في اللغات الأخـري يُمكِننـا أن نَري أنهما ا ستهلكا مجهو داً قريبا لدعمهما. كما أن لهما تقريبـاً ذات الدرجـة مـن الإضـافات لقواعـد اللغات اللاتي يُضمَّان إليهن !

فلِمَ بُذِل ذلك الجهد و خُصِّصَت تلك القواعد لدعم الواجهات و لـم يُبْذَلا و يُخصَّصَا لـدعم الوراثـة المُتعددة ؟!

السبب تَجدونه عند أ صحاب توجه دعم الواجِهات بدلًا من الوراثة المُتعددة.

- **تُعطِى الصِنف أو الأصناف خصائص ما كانت لتمتلكها:**

قَرأتُ هذا السبب فى مرجع إنقليزيٍ يتحدث عن لغة الـjava، و فى الحقيقة فـإننى لا أفهـم حـتى الآن ما يقصده المؤلف من هـذاً الكلام. فـإن كـان يقصـد ميـزة إعطـاء الهياكـل و الأ صـناف ميزتـى الوراثـة و الوراثة المتعددة على الترتيب فقد ر ددنا على هذا الزعم.

- **إجبار الصنف الجديد البانى لها على بناء كل الإجراءات التى فيها:**

و أنا أتساءل: لِمَ هذا التكلف ما دامت هنـاك الأ صـناف المُجـرَّدة الـتى تـُؤدِّي هـذا الـدور بكـل قـوةٍ و فعالية ؟!

- **تُمَثِّل دور البروتوكول بين الأصناف المختلفة، و لذلك فبعض لغـات البرمجـة الكائنيـة التوجه تسميها (البروتوكول) بدلًا من الواجهة:**

يُمكِن للأ صناف المُجرَّدة أن تقوم بنفس الدور بمنتهى الكفاءة، و أُكرِّر أن هذا القول يشترك مـع سـابقه فى أن مُصمِّمي اللغات التى تدعم الواجِهات إختاروا الحل الأ صعب لمُشكِلةٍ كان يُمكِن حلهـا بمنتهـى البساطة !

الهَياكِل structs:

نبذةٌ بسيطة:

هي مُكوّناتٌ في الـC و الـ++C و الــ#C تُشبِه إلى حدٍ كبيرٍ للغاية الأ صناف، حيث تُمثّل تجميعةً منطقيةً من المُتغيّرات، بالإضافة إلي الإجراءات التي تعمل على تلك المتغيرات. و على الرغم من هذا فهي تُخالِف الأ صناف في بعض الخصائص كما سأوضّح فيما يلي.

أري ألا يُسمَح بها للأ سباب التالية:

- تُدخِل المُبرمِج في تفا صيلٍ يُفتَرض أن تظل مقصورةً على المُبرمِج الباني للمُترجِم و لا تُتاح لمُستخدِم اللغة؛ فالسبب الرئيس لدعمها في لغاتٍ مثل الـC و الـ++C أنها تُعطِي المُبرمِج القدرة على تحديد ما إذا كان يُريد إنشاء الكائن الجديد في منطقة المُكدّس stack في الذاكرة أم في منطقة الـheap، و كما قلنا أن هذا يُعَدُّ تعدياً واضحاً على السلطات التي يُفتَرَض أن تكون للمُترجِم أو المُفَسِر مع نظام التشغيل.

فمُتَرجِمٌ مُهَنْدَسٌ جيداً و مُبَرْمَجٌ على مُستوَىً عالٍ من الحِرَفية و الإتقان يُغنِي عن كل هذه الزيا دات في القواعد، و يُغنِي المُبرمِج عن التدخل في أمورٍ يشتكى أغلبية المبرمجين من اضطرارهم إلى خوض غمارها رغم عدم رغبتهم في أغلبية الوقت إن لم يكن كله في فعل هذا.
و القول بأن هناك من المُبرمِجين من تُسعده هذه الخا صية في اللغات التي تدعمها تجعلني أقترح على هذا المُبرمِج العو دة للبرمجة بلغات التجميع، فالأخيرة ستُعطِيه من القوة الأكثر و سيجد مُبرمِجين يقولون أن البرمجة بها تُسعدهم أكثر من سواها !

أما ما أظنُه السببَ وراء وجو د هذه الصفات في اللغات التي تُوجَد بها: فهو أن ظروف خروج لغةٍ مثل الـC كان هو السبب الرئيس وراء كل ما حدث؛ حيث أن هذه اللغة قد أنتجها دِينِس رِيتشِي Dennis Ritchie حين احتاج إلى لغةٍ تُعطِيه كل القوة التي تُعطيها له لغات التجميع (أو على الأقل أغلبها)، في حين تكون لها بساطة لغات المستوى الأعلى و **محموليتها** و قابلية أكوا دها العيش لفترةٍ أطول و الإ ستفا دة المتجد دة من التِقنيات الجديدة في برمجة المترجمات، و ذلك ليكتب بها نظام التشغيل الجديد الذي يرغب في بنائه، و هو نظام اليُونِكس unix.
و كان أمامه خِياران لأ سلوب التعامل مع القوة التي تُتِيحها له اللغة: **فالأ سلوب الأول** و هو الأكثر مُبا شرةً كان أن يضع كل تلك القوة في يديه هو على هيئة أ دواتٍ واضحةٍ يَستخدمها كما يَستخدم المُتغيرات و الإجراءات، أما **الأ سلوب الثاني** فهو أن يضع كل تلك القوة في بِنية

المُترجِم ذاته ثـم يُكيفهـا لتُنتِـج لـه المنتجـات الأفضـل تقنيـا، بخُوارِزْمـاتٍ ذكيـةٍ يُمكِنهـا أن تَسـتغل الفرص التحسينية في الاكواد و تقوم بجعل الكود أفضل أداءاً و استغلالا للموارد.

و لأن دِينِس كان مُعتاداً على البرمجة باستخدام الأسلوب الأول (و هـو تَـوافر كـل الأدوات بيـن يديه)؛ لأنه عاصَر كل الأجيال الأولى للغات البرمجة منذ لغة الآلة و حتى اللغات عالية المسـتوى مُروراً بلغات التجميع، و لأن الأسلوب الأول هو الأسلوب الأسرع فى البناء: فقـد وقـع اختيـاره عليه.

و لما كان مُعاصِروه لهم نفس ظروفه التى دفعته إلى اختيـار ذلـك الأسـلوب فقـد وافقـوه أو علـى الأقل وافقه معظمهم على اختياره هذا.

و لهذا كله فإن الغلطة لم تكن هنا، بل كانت فى اللحظة التى كـان بإمكـان المُـبرمِجين مـن الأجيـال التالية الإستغناء عن هذا الأسلوب و العودة إلى الأسلوب الأفضل و لكنهم لم يفعلوا، و كان علـى رأس هذه اللحظات لحظة إنتاج لغة الـ++C؛ فقد أدخلت هذه اللغة إمكانيـات البرمجـة كائنيـة المَنحَى إلى لغة الـC و لكنهـا احتفظـت فـى نفـس الـوقت بكـل عيـوب الأخيـرة الـتى نَجمـت عـن الأسلوب الأسرع فى التحكم فى القوة البرمجية، و لم يُحاوِل مُطوّرها التخلص مـن تلـك العيـوب (بل تَوسَّع أكثر من ذلك فى وضع مهامٍ كان يُفتَرض على المُترجِم القيام بها علـى كاهـل المُبرمِج العادى !).

- تزيد من قواعد اللغة بدون فائدة، بل إن الشبه بين قواعدها و قواعد الأصناف يجعلهـا مصـدر بلبلـةٍ للمُبرمِج، و أكثر للمُتعلِّمين الجدد للُّغة التى تدعمها؛ فعلى المُبرمِج أن يتذكر أن الهيكل لا يَرث و لا يُورَث على العكس من الصِـنف الـذى يَـرث و يُـورَث، و عليـه أن يضـع فـى ذهنـه أن الهيكـل و الصنف يشتركان فقط فى الوراثة من الواجهات، هذا بالطبع بجانب تذكر أن الهيكل حينما يُنشأ فإنه يُنشأ فى منطقة المُكدِّس فـى الـذاكرة بينمـا يُوضَـع الصِـنف فـى منطقـة الــheap. و غيرهـن مـن الإختلافـات الـتى لا سـبيل للتخلـص منهـا إلا بـالتخلص مـن التفرقـة بيـن الصـنف و الهيكـل مـن الأصل.

فإذا ما ضَممنا الهيكل و الواجهة إلى بعضهما البعض لوجدنا أنهما ليسا إلا تنويعـاً علـى الصِـنف، و لكن مع بعض الإختلافات: فى الوظيفة كما فى حالة الواجهة، أو فى الخصائص الداخلية كمـا فـى حالة الهيكل.

و قد كان بإمكان مُطوّري لغات البرمجة الجديدة إن أرادوا دَمْج هذه الأنواع الثلاثـة مـن المُكوّنـات فى مُكوّنٍ واحدٍ أشـمل و أعـم (كمـا فعلـتُ فـي إبداع) فيتفـادوا كـلٍ ذلـك الزَّخـم مـن القواعـد المُربِكة، و كذا كل البلبلة التى تَنتُج عن التشابه بين تلك القواعد نظرياً و اختلاف التعامـل العملـى معها.

و لكن ما حدث هو ما نَوّه إليه البروفيسور C. A. R. HOARE فى ورقةٍ علميةٍ ماتِعَةٍ تحت عنوان (ملاحظاتٍ حول تصميم لغة البرمجة) قَدّمها عام 1973، حيث قال فيها ما ترجمته (ببعض التصرف):

> بعض مُصمِّمى لغات البرمجة استبدلوا (النمذجة) بـ(البساطة) فى تصميم اللغات، و يعنون بـ(النمذجة) أن المُبرمِج إذا لم يستطع فهم اللغة كاملةً فإنه يكفيه فهم جزءٍ منها فقط.
>
> ربما يكون هذا معقولاً بالنسبة للبرامج التى تعمل كما قصد منها المُبرمِج، و لكن إذا كان برنامجه لا يعمل و كان (قَدَرًا) يستخدم صفةً فى اللغة لا يعلمها المُبرمِج: فإنه سيقع فى مشاكلٍ صعبة. فإذا ما كان مُوَفَّقًا نوعاً ما فإن البرنامج المُصَحِّح سيتحقق من خطئه و لكنه لن يكون قادراً على فهم الرسالة التشخيصية، خلاف ذلك فإنه لن يكون ذى فائدة.
>
> و إذا ما أُضِيف إلى تعقيد لغة البرمجة تعقيدُ بنائها، و تعقيدُ بيئة عملها و كذا تعقيدُ معايير العمل لاستخدامها: فإنه ليس من المُفاجِئ أنه عند مجابهة مهمةٍ برمجيةٍ معقدةٍ فإن العديد من المُبرمِجين لا يستطيعون النجاة.

فقد اعتقد مُطوِّروا مُعظم اللغات الحالية أنه بإمكان المُبرمِج إتقان جزءٍ فقط من اللغة و عدم الحاجة إلى إتقانها كلها، و لكن الرد عليهم ما قاله البروفيسور HOARE و أدرجناه فيما سبق.

جُمْلَة goto :

من أشد الأمور التى تُحيّرنى فى عالم تصميم لغات البرمجة أن هناك أموراً معينةً ظَل كثيرٌ من العلماء و المُحترفين من المُبرمِجين يُنوّهون إلى خطرها و يُحذّرون منها بشدة، ثم رغم هذا يقع فيها مُصمِّمو اللغات بكل بساطة !، و من أشد الأمثلة وُضوحاً مسألة جُمْلة goto التى اختصها بعض العلماء و هو edsger w.dijkstra برسالةٍ علميةٍ خاصةٍ للتحذير منها و من خطرها، و هى الرسالة المُسَمّاة (جملة goto تعتبر ضارة)، بل و قد اشتهر بين المُبرمِجين مُصطَلح (معكرونة الاسباقيتى) ليصفوا به البرامج التى تُفرِط فى استخدام جملة goto فتتحول إلى عجينةٍ من الأوامر التى يصعب فهم عملها حتى على مُبرمِجها إن غاب عنها لحظات !

و قِيل عن فائدة عدم وجود جملة (goto) فى البرنامج[8]:

و إذا كان بالإمكان قراءة البرنامج من أعلى إلى أسفل بدون الحاجة إلى القفز عبر الجُمَل المتناثرة نتيجةً لاستخدام جُمَل (goto) المُربِكة: فإن التركيز سيكون أكبر لأنه سينصب عندئذٍ على جزء الكود الحالى، مما يُعطِى الفرصة لاستغلال الذاكرة اللحظية جيداً و عدم تكليفها متابعة أجزاءٍ أخرى من الكود بطريقةٍ تُشَتّتها و تُهدِر قواها. و هو نفس السبب الذى يجعل المُبرمِجين الذين لا يستخدمون هذه الجُمَل فى برامجهم أقل عرضةً للوقوع فى الخطأ، و نتيجةً لهذا فقد نجحت البرمجة الهيكلية فى السبعينات، حيث أنها تقوم على البرمجة فقط باستخدام الجُمَل الشرطية و الجُمَل التكرارية مما يجعل البرامج أقل أخطاءاً.

و فى الورقة العلمية المُعَنْوَنَة (جملة goto تعتبر ضارة) نَوّه المؤلف (edsger w.dijkstra) إلى أن جملة goto دعوةٌ صريحةٌ للإرتباك و البلبلة، حيث قال ما ترجمته (ببعض التَصَرُّف):

الاستخدام الجامح لعبارة Goto له عواقبٌ فوريةٌ من حيث أنه يُصبِح فى غاية الصعوبة إيجاد مجموعة إحداثياتٍ ذات معنىً يُمكِننا من خلالها توصيف العملية. عادةً فإن الناس يأخذون فى اعتبارهم كذلك قيم متغيراتٍ مُختارةٍ بعناية، لكن هذا خارجٌ عن نطاق التساؤل لأنه لا يُمكِن فهم معنى هذه القيم إلا بالنظر إلى التقدم ! مع جملة Goto يُمكِن للمرء توصيف البرنامج بشكلٍ فريدٍ بعَدّادٍ يُحصى عدد الأحداث التى نُفّذت منذ بداية البرنامج. الصعوبة أنه فى إحداثيٍ كهذا على الرغم من تفرده أنه ليس مفيداً على الإطلاق. فى نظامٍ إحداثيٍ كهذا يُصبِح من الأمور المعقدة للغاية تعريف كل تلك النقاط للتقدم.

8 مـن كتاب (software engineering) الفصل الثاني (human factors in software engineering)، أما المُؤلّف فلم أستطع الحصول علي اسمه بكل أسف !؛ فالنسخة التي عندي ليس فيها غلاف الكتاب، كما أن البحث عنه علي الشبكة لم يُجدِ نفعا.

و قال كذلك ما ترجمته:

جملة Goto كما هو الوضع حالياً بدائيةٌ للغاية، و تُعَد بصورةٍ كبيرةٍ دعوةً لتحويل البرنامج إلى فوضى.

و من ثم كان من الطبعى أن ينتهى به الأمر إلى النتيجة التى ذكرها فى بداية رسالته، حيث قال ما ترجمته:

لعديدٍ من السنوات كنتُ مُعتاداً على مُلاحظة أن كفاءة المبرمجين تُعتَبَر دالةً عكسيةً فى كثافة عبارة Goto فى البرامج التى يُنتِجونها، و فى توقيتٍ أحدث اكتشفتُ لم لاستخدام Goto هذه التأثيرات الكارثية، و اقتنعتُ بأن جملة Goto يجب أن تُمحَى من لغات برمجة المستوى الأعلى (أى: كل شئ فيماعدا شفرة الآلة المبسوطة).

و على الرغم من أن هذه الرسالة نُشِرت فى مارس من عام 1968 م، و صَدَر بعدها مُؤلَّفاتٌ أخرى تُحذِّر مثلها من المشاكل التى تُسبِّبها جملة goto: إلا أن مُصمِّمي لغات البرمجة ظلوا بعد كل تلك السنين على عنادهم، و أبقوها فى اللغات الجديدة التى كان يجب فيها التخلص من كل ما عُلِم خطره على المُبرمِج.

و لا يُمكِننى بالضبط أن أحدد السبب الذى يجعل لغات برمجةٍ كائنية التوجه إلى أقصى الحدود مثل الـ C# تُبْقِى على جُمْلةٍ أَمْرِيةٍ من العهود الأقدم للغات البرمجة، على الرغم من أن وجودها إضافةً إلى ما يُسببه من ارتباكٍ منطقي فى البرامج فهو أيضاً يُسبب زيادةً فى قواعد اللغة التى كانت تحتاج أصلاً إلى التقليل من كميتها المتورمة لا زيادتها !

المُؤشِّرات pointers:

نبذةٌ بسيطة:

تُوجَد فى عائلة الـC إلا الـjava و في العديد من اللغات الأخري مثل الـobject pascal و غيرهن، و هى مُتغيراتٌ من نوعٍ خاص؛ حيث أن القِيَم التى تَحمِلها داخلها تُمثِّل عناويناً فى الذاكرة إما لمُتغيرٍ آخرٍ أو لبداية جُمْلةٍ نصيةٍ أو حتى لإجراءٍ مِن الإجراءات، و حينما يتم التعامل معها فإن المُبرمِج يتعامل مباشرةً مع الذاكرة و مُحتوياتها دون أى وسيطٍ منطقي. و التعامل بالمؤشرات يكاد يكون إجبارياً فى الـC/++C اللاتى يُدخِلنها فى أى أمرٍ مهما كان صغيراً (و أقول هذا عن تجربةٍ شخصية).

أري ألا يُسمَح بها للأسباب التالية:

- تُدخِل المُبرمِج فى تفاصيلٍ يُفترَض أن تظل مقصورةً على المُبرمِج البانى للمُترجِم لا مُستخدِم اللغة، حيث أن إدارة الذاكرة و إعادة غير المُستخدَم منها مرةً أخرى إلى حوزة نظام التشغيل من صميم عمل المُترجِم و مُبرمِجه، و الذى يجب ألا يتخلى عن هذه المهمة لصالح المُبرمِج العادى أبداً.

و لكن كما قلنا قبلاً عند مناقشة أسباب التخلى عن استخدام الهياكل فى لغة إبداع فإن المُبرمِجين القُدَامى اختاروا أسرع الحلول للحصول على القوة القصوى فى برامجهم، و قد كان بكل أسفٍ الحل الأقذر (و القذارة هنا بالطبع معنويةٌ بحتة)؛ فقد كلَّف هذا الحل المُبرمِجين الجدد الكثير: فبدلاً من الإنطلاق نحو مزيدٍ من التركيز على المهام التى تُستَخدَم لغات البرمجة لحلها فقد أصبح المُبرمِج يَقضى كثيراً من وقته أمام اللغة مُحاوِلاً الهروب من الحُفر المُغطاة و الألغام المُختفية؛ للحصول على حقه الطبعى فى برنامجٍ خالٍ من العيوب و يعمل بكفاءةٍ و قوة !

و أَصبحَ من المُعتاد بالنسبة للدارسين الجدد تنبيههم على النقاط التى يجب دراستها بمُنتهَى الحرص و التركيز، و أصبح من الأكثر اعتياداً طمأنة هؤلاء الجدد حينما يُخطِئون بسبب تلك الألغام بالقول لهم أن **"هذا يحدث حتى للمُبرمِجين الكبار"** !، و كأن هذا شئٌ يبعث على الطُمأنينة لا الذعر !

و لى هنا أن أُدرِج قولاً قرأتُه فى مجلة (مُجتَمع لينوكس العربى) للأخ (محمد نجم):

> عندما اتذكر بداياتي في البرمجة .....
> يسعني القول أنها كانت مأساة حقيقة فانا شققت طريقي في البرمجة بلغات معقدة كالسي وبعض الجافا ولي أسبابي فلم اكن اعلم شيء أخر سواهم وكنت صغير السن وقتها وغير متفرغ أيضاً .خليط الإحساس والإحباط ثم الغباء ثم الغضب ثم تنهي برنامج بسيط(من ٤٠ سطر او اكثر) وتجد أخطاء وتعجز عن إيجاد البعض الآخر هذا بالفعل كفيل بجعلي ارجع عن البرمجة ككل وربما ابدأ في البحث عن مجال أخر بعيداً عن هذا التعب(وجع الرأس).

و لتكتمل الصورة فإنى أُدرِج قوله عن البرمجة بلُغاتٍ أخرى:

أنا بدأت بروبي حقيقة بعد العديد من المشاكل (وجع الراس) مع السي والجافا والشيل سكربتينج (الذي كان اسهلهم لكنه لايفي بالغرض) والان انا مستمتع ببساطتها وقوتها معاً اذا اردت برنامج ويب لدي Railsواذا اردت برنامج رسومي توجد العديد من المكتبات تفي بالغرض

و الأمر واضحٌ تماما، و يعكس بالفعل ما يحدث لكثيرٍ من المُبتدِئين فى البرمجة و يُسبب خسارةً كبيرةً فى نشء المُبرمِجين و المُقبِلين على دراسة هذا العلم.

- لا تكتفى بكل تلك البلبلة التى تُسبِبها بل تزيد من قواعد اللغة بشكلٍ كبير، فتُصبِح عملية تعلم اللغة أعقد و أطول و أكثر إرهاقاً عما كانت ستُصبِح عليه لو لم تكن تلك المُكوّنات موجودةً فيها.
و لا تكتفى بإرهاق مُستخدِم اللغة فقط، بل تتجاوز ذلك إلى عِقاب المُصمِّم لها بطريقةٍ لو علم بها مُستخدِم اللغة لانتشى شماتةً و نام قرير العين؛ فالمُصمِّم الذى يجعل لغته البرمجية كبيرةً إلى حدٍ مُفزِع ليس له أن يَشتكى من صعوبة التأليف بين كل ذلك الزخم من القواعد، و ليس له كذلك أن يَشتكى من صعوبة صُنع المُترجِم أو المُفسِّر القوى لها.

أما تطوير اللغة و الإرتقاء بها تدريجياً فأمرٌ سيكون بالنسبة له ضرباً من ضروب الخيال العلمى؛ فإذا كان حال اللغة و هى فى أول إصداراتها على تلك الحال المُزرِية التى و صفنا: فماذا سيكون حالها عند إضافة الجديد إليها ؟!
إضافةً إلى ذلك فحتى لو استطاع إضافة الجديد إليها فأى مُبرمِج مجنونٍ ذلك الذى سيتحمل كل تلك الإضافات و يتماشى معها ؟!
أما المساكين ممن فَرَضتْ عليهم ظروفهم الدراسية تعلُّم تلك اللغة البرمجية فلن يجدوا مفراً بعد ذلك من تكملة دراستها كاملةً و احترافها؛ فعلى الرغم منهم سيكونون قد قطعوا شوطاً لا بأس به فى تعلم اللغة و لن تكون خسارتهم عند تكملة دراستها أكبر من خسارتهم عند تركها نهائياً و تعلم لغةٍ جديدةٍ تماماً، بل على الأرجح ستكون تلك الخسائر أقل و أهون و هو الأمر الوحيد الذى يجعلهم يُتِمون مشوار التعلم هكذا (إن أتموه).

- مصدرٌ لا ينفد للأخطاء النحوية و التنفيذية و المنطقية فى البرامج التى تُستَخدم فيها، و بالتالى فإن المُبرمِج للمُترجِم حينما يُلقِى بجزءٍ من مسئولية إدارة الذاكرة على عاتق المستخدم لتلك اللغة فإنه لا يُقلِّل الأعباء التى تقع عليه كما يظن، بل إنه يزيدها أكثر و أكثر لحاجته إلى اكتشاف أنواعٍ أكثر من الأخطاء و المساعدة على تنقيح البرامج حتى تخرج سليمة النحو و المنطق.

و كون المؤشرات مصدراً للتصعيب فى البرمجة لا يخفى على أحد؛ فهى أدواتٌ مُركبَةٌ للغاية بطبيعتها. و عن تجربةٍ شخصيةٍ أقول أن وجود المؤشرات فى البرامج التى أكتبها فى عملى كانت و لا تزال تزيد من صعوبة التكويد إلى الحد الأقصى مما يُكلِّفنا من الوقت الغالى الكثير.

و أَضرِبُ مثالًا لهذا الموقف بالكود التالى لدالةٍ تتعامل مع مكتبة الـOpenCV حيث تأخذ منها متغيراً من نوع (IplImage *) حيث يُمثِّل هذا النوع التمثيل القياسى للصورة فى مكتبة الـOpenCV لتُعيد ما يحتويه من بيانات الصورة على هيئة بايتاتٍ مُتراصةٍ يُشِير إلى أولها مُؤشرٌ من نوع (double *). و المُفترَض أنها مسألةٌ بسيطةٌ للغاية و مُباشِرةٌ و لكن استخدام المؤشرات فيها حَوّل الأمر إلى مأساة.

حيث أن التمثيل IplImage يستخدم مُؤشِّراً من نوع (Char *) ليُشِير إلى أول بايتٍ فى بيانات الصورة التى يحتوى عليها. و يُسَمَّى هذا المؤشر بـimageData. و من ثَمَّ للحصول على هذه البيانات و التعامل معها بأى شكلٍ من الأشكال يجب على المُبرمِج التعامل مع المُؤشِّرات بكل ما يحمله هذا من معانى.

و بالنظر إلى كود الدالة cvtImg2Dta و المقارنة بين كل ذلك الزخم من الحسابات المُستَخدَم للوصول إلى بيانات الصورة فى حالتنا هذه. و بين ما كان الوضع ليُصبح عليه إذا كانت البيانات موجودة على هيئة مصفوفة ثنائية الأبعاد two dimensional array: لوجدنا أن الوضع كان ليُصبح أسهل بكثيرٍ و أقل تعقيداً مما أجبرتنا عليه المؤشرات بالفعل.

```
void CvtImg2Dta(IplImage *img, double *slice)
{
    for( int y=0; y< img->height; y++ )// taking care of rows indexing
    {
        for( int x=0; x< img->width; x++ )// taking care of columns indexing
        {
            slice[img->nChannels*x+y*img->width*img->nChannels] =
                    (double)img->imageData[img->nChannels*x+y*img->widthStep]/255.0;
            if(img->nChannels==3)
            {
                slice[(3*x)+(3*y*img->width) +1] =
                    (double)img->imageData[(3*x)+(y*img->widthStep) +1]/255.0;
                slice[(3*x)+(3*y*img->width) +2] =
                    (double)img->imageData[(3*x)+(y*img->widthStep) +2]/255.0;
            }
        }
    }
}
```

فبدلًا من كتابة:

```
( (3*x) + (3*y*img->width) +1 )
```

و ما شابهها من حساباتٍ كنا سنكتب مُمَيّزات العنصر element indexes داخل المصفوفة مبا شرةً و ينتهى الأمر.

و ليس الأمر بالبسيط كما قد يبدو؛ فقد تَسببتْ كل هذه الحسابات المُتداخِلة المُربِكة فى تعطيل عملنا لفترةٍ لا بأس بها؛ بسبب وجود خطأٍ ما فى عمل الدالة احترنا فى سببه، حتى اكتشفنا أننا نسينا مُعامِلًا مُعيّنًا من ضِمن الحسابات المُربِكة تَسَبّب عدم وجوده إلى عدم الإشارة بصورةٍ صحيحةٍ إلى العنا صر المرغوب فيها !، و لا تَسَل عن الحسرة و الغيظ اللذين أحسسنا بهما حينما اكتشفنا أن ذلك الخطأ التافه هو السبب فى كل الربكة و الحيرة اللتان غمرتانا، و كل التأخر الذى حدث لعملنا.

لذلك أقول عن جَدٍ و عن تجربةٍ شخصيةٍ واقعيةٍ أن الإصرار على وجود المُؤشِّرات فى لغات البرمجة الجديدة لا يخدم العلم البرمجى فى شئ، و لا يُقدِّم الخير للأجيال الجديدة من المبرمجين الذين يُفتَرض بهم السير بهذا العلم الجليل إلى الأمام، و هم بطبيعة الأمر لن يفعلوا هذا إذا ما أ صرّت الأجيال الأقدم من المبرمجين على فرض تَركتهم الثقيلة من الأدوات العتيقة و الأفكار المُربِكة التى ولّى زمنها و لم يعد من المنا سب بحالٍ من الأحوال أن تبقى فى الواقع المَعِيش للعلم البرمجى.

الإجراءات procedures:

نبذةٌ بسيطة:

الإجراءات التي أقصدها هنا ليست هي الإجراءات التي في إبداع بالطبع، بل هى مُكوّناتٌ قديمةٌ للغاية حيث يرجع ميلادها إلى زمن الأيام الأولى للبرمجة عالية المستوى، و هى تُشبه الدوال functions تماماً فى الفكرة الأساسية من حيث أنها تُمثِّل عباراتٍ من الكود تُنَفَّذ بالترتيب من أعلي إلي أسفل فى كل مرةٍ يتم فيها استدعاء ذلك الإجراء، و تختلف عن الدوال فى أن الدوال تُعِيد قيمة واحدةً أو قيماً متعددة أما الإجراءات فلا تُعِيد أى قيم.

و من اللغات التى تَدعم الإجراءات فى الوقت الحالى: الـFortran و الـAda و الـPascal و هى كلها لغاتٌ من الأجيال القديمة، و فى مُعظم اللغات البرمجية الحديثة تم الإستغناء عن الإجراءات بشكلٍ نهائيٍ حيث لم يَعد لها أى معنى (اللهم إلا في لغاتٍ مُعيَّنةٍ مثل الـobject pascal)، و إنى أوافق مُصمِّمى اللغات الحديثة فى قرار استبعاد الإجراءات كمُكوّنٍ فى لغات البرمجة الحديثة للأسباب التالية الذكر.

أري ألا يُسمَح بها للأسباب التالية:

هناك سببٌ وحيدٌ لعدم دعم الإجراءات فى لغات البرمجة الحديثة هو أوضح من الشمس فى مُنتصَف السماء، أن الإجراء ما هو إلا دالةٌ لا تُعِيد أى قيمة، و لا يختلف عن الدوال العادية فى أى أمرٍ آخرٍ سوى هذا، و بالتالي فأى لغة برمجةٍ تدعم الدوال (أى كل لغات البرمجة الحديثة بطبيعة الحال) تدعم الإجراءات ضمنياً لكن مع عدم ذكْر كلمة إجراء بشكلٍ صريح.

و لكن على الرغمِ من أن هذا الأمر بَدَهيٌ تماماً و من غير المنطقى تطبيق غيره فإن هناك لغات برمجةٍ سارت عكس التيار تماماً و دعمت الدوال و الإجراءات كلٌ بشكلٍ مُنفَصِلٍ عن الآخر !، و هنا سأتكلم عن لغة الـ object Pascal بعينها لمجرد ضرب المثل، ففيها يكون تعريف الإجراء كما فى المثال التالى:

```
Procedure foo (var a: byte);
Begin
  Writeln (a);
  a: =10;
  Writeln (a);
End;
```

حيث أن الكلمات (Procedure) و (Begin) (End;) هى كلماتٌ محجوزةٌ ثابتةٌ يجب كتابتها فى كل مرة، بينما الكلمة (foo) هى ا سم الإجراء الذى يُمكِن تغييره بحرية، و ما بين القوسين (التاليين للاسم) هى قائمةٌ بمُعامِلات الإجراء التى ستُمَرَّر له للعمل عليها، و يُوضَع الكود المُكَوِّن لجسم الإجراء بين كلمتى (Begin) و (End;) المفتاحيتين.

و لو نظرنا إلى المثال التالى لتعريف دالةٍ من الدوال فى الـobject Pascal:

```
Function w (a, b: integer): integer;
Begin
Read (a, b) ;
W: = a+b;
End;
```

لوَجَدنا أن نفس القواعد السابقة تُطَبَّق هنا أيضاً فيما عدا فَرْقَيْن اثنين لا ثالث لهما:

- كتابة الكلمة المفتاحية (Function) بدلاً من الكلمة (Procedure) فى بداية جُمْلة التعريف كما هو مُوضَّحٌ بالمِثالين. و هذا الفارق هزلىٌ كما هو واضحٌ و لا يعنى أى شئ.
- كتابة نوع الخَرْج الخاص بالدالة بعد أقواس المُدْخَلات المُمرَّرة لها، و هو من النوع (Integer) فى المثال السابق.

و فيما عدا هذين الفارقين فالأمر واحدٌ لا خلاف فيه.

و ما دام الأصل واحدٌ، و ما دامت الدالةُ تحتوى الإجراءَ داخلها بشكلٍ منطقي: فإن الأمر المثير للعجب و السخرية هو أن تدعم لغة برمجةٍ المُكوّنين فى نفس الوقت ! ؛ فلا هى فَعَلت مثل اللغات الأقدم التى لم يكن هناك ما يُسمَّى بالدوال فى وقتها و بالتالى فَعلت الصواب حينما دَعَمت الإجراءات؛ لأنها دَعَمت مُكوّناً قوياً بمعايير عصرها، و لا هى فَعلت مثل لغات البرمجة الحديثة التى اكتفت بالمُكوّن الأعم و الأشمل و الأقوى الذى هو الإجراء؛ و بالتالى فعلت الصواب بتقليل قواعد اللغة و تخفيف اللَّبس و البلبلة من على كاهِل المُبرمِجين.

و السبب الوحيد الذي يُمكِن أن يُقدّمه مُصمّمو تلك اللغات أنهم أرادوا من وراء ذلك المُحافظةَ علي التوافق العكسي backward compatibility؛ لأن تلك اللغات الحديثة هي في حقيقة الأمر تطويرٌ للغاتٍ أقدم، فمثلاً تُعَد لغة الـobject Pascal تطويراً للغة الـPascal التي تحتوي بدورها علي الإجراءات، و لما كانت برمجياتٌ كثيرةٌ مكتوبةً بالـPascal فإنهم اعتبروا أنه من الخسارة تَرْك كل

تلك البرمجيات و عدم دعم ترجمتها بشكلٍ طبعيٍ في لغتهم الجديدة، و نفس الحال تكرر مع لغتي الـ++C/C.

و لكني حتي في هذه الحالة أَرفضُ ضم الإجراءات؛ لرفضي لمبدأ التوافق العكسي أصلًا؛ فلغات البرمجة الحديثة يجب أن تنفصل عن القديم لتُقدِّم ما يتوافق مع أحوال الواقع البرمجي الحالي.

و يُمكِن التغلب علي عدم التوافق بين برامج اللغتين القديمة و الحديثة عن طريق بناء برنامج لتحويل الأكواد بينهما آلياً، و هو الأمر الموجود بالفعل للغاتٍ تختلف بشدةٍ عن بعضها البعض (مثلً الـjava و الـ objective C) فبالتالي سيكون أسهل بمراحلٍ عند عمله للتحويل بين لغتين شديدتي التشابه مثل الـ object Pascal و الـPascal.

الوَسائِط delegates:

نبذةٌ بسيطة:

الوسائط (و هنا سأُركِّز علي الوسائط كما تُوجَد في لغة الـ#C) هي عبارة عن مُكوّناتٍ من نوع خاص تسمح بتسجيل استدعاءاتٍ عامّةٍ لدوالٍ لها مُعامِلاتٌ واحدةٌ و أنواع خَرْجٍ واحدةٌ كذلك، بحيث أنّه يُمكن ربط الوسيط الواحد مع أكثر من دالة، و حينما يتم استدعاء ذلك الوسيط (كما تُستَدعى أية دالةٍ أخرى) يَقومُ الوسيط باستدعاء الدوال المرتبطة به، و يُمَرّر لها المُعامِلات التى تم تمريرها له عند استدعائه.

و لا يَمنع التشابه بين مُؤشِّرات الدوال functions pointers و الوَسائط من كَوْن الوسائط نوعاً جديداً؛ حيث أن مُؤشِّرات الدوال لا يُمكِنها أن تقوم بنداء أكثر من دالةٍ فى نفس الوقت و تقتصر فى عملها على استدعاء دالةٍ واحدة، و بالتالى تكون الفائدة المرجوة من ورائها صغيرةً نوعاً ما. أما الوسائط فيُمكِن لها (على الأخص تلك التى يُمكِنها أن تشير إلى دوالٍ لا تُعيد أى قيمة) أن تشير إلى أكثر من دالة.

أري ألا يُسمَح بها للأسباب التالية:

الوسائط (من حيث الفكرة الأساسية) مُكوّناتٌ فى غاية القوة و الفائدة، و لكن المُشكِلة أن الشكل الذى قدّمته لغة الـ#C للوسائط كان فى غاية الهُزَال و الضعف.

و ضعْف صياغات الوسائط فى الـ#C يتجلى فى الصور التالية:

- تقييد المُعامِلات التى ستُمَرّر للدوال التى سيستدعيها الوسيط بأن تكون هى نفسها المُعامِلات التى تُمرّر للوسيط ذاته، و بالتالي لا يُمكِنني اختيار تمرير مُعامِلاتٍ مُختلفةٍ حتي لو كانت هناك حاجةٌ ماسّة لذلك. و هذا العيب أدى إلى عيبٍ آخرٍ هو:

- يجب أن تكون مُعامِلات الدالة التى تُضَم إلى الوسيط مُوافِقةً لمُعامِلات الوسيط، أى أن الوسيط يُشير إلى نوعٍ واحدٍ من الدوال (هذا بالإضافة إلى التقيد بنوع الخرج المُوَحّد بين الوسيط و إجراءاته). و هكذا فقَدَ الوسيط ميزة العُمومِيّة التى تُتيح له أن يُشير إلى أى نوعٍ من أنواع الدوال مهما كانت مُعامِلاتها و خَرْجها.

المهمات tasks:

نبذةٌ بسيطة:

المهمة task هى مُكوّنٌ برمجىٌ قائمٌ بذاته فى لغة الـAda، و هى فى الأ صل برنامجٌ يُشبِه البرنامج الأ سا سى فى كل شيئ إلا أنه يتم تنفيذه على التوازى مع البرنامج الرئيس. أى أنها تقوم بنفس عمل خيوط التنفيذ execution threads فى اللغات الأخري و لكن بشكلٍ أكثر اندماجاً داخل اللغة نفسها و ليس بشكلٍ مُنفصلٍ عنها فى مكتبة اللغة.

أري ألا يُسمَح بها للأ سباب التالية:

فى الحقيقة فإننى لا أعترض على المهمات كمُكوّنٍ أ سا سيٍ فى لغات البرمجة الحقيقية و لا أُجا دِل فى هذا على الإطلاق؛ فأنا من أكثر النا س اقتناعاً بأهمية دعم العمليات المتوازية التى تُركّز التقنيات الحديثة فى تصميم المُعالِجات على دعمها بتعديد أنوية تلك المعالجات فى الشريحة الواحدة و غيرها من التقنيات الأخري.

و لكننى أرى فقط أنه من الأفضل عدم وضع هذا المُكوّن فى صلب اللغة مبا شرةً و وضعه فى المكتبة القيا سية الخا صة بها قدر الإمكان. فمثلاً فى الـjava و الـ#C يُمكِنك إذا ما احتجت إلي خيوط التنفيذ المُتعددة أن تمد يدك إلى المكتبة و تُعرّف خيط تنفيذٍ جديد، و تربط الخيط الجديد إما بصنفٍ جديدٍ يبنى implements واجهةً interface مُعيّنةً كما فى الـjava (و هو نموذجٌ مُتزمّتٌ بعض الشيئ في وجهة نظري)، أو تربطه با سم دالةٍ ما كما فى الـ#C (و أظنُ أنه النموذج الأفضل و الأقوى و الأ سرع).

و لكن يظل الأمر سيان بالنسبة للمُستخدِم النهائى للُّغة؛ حيث ستظل المهمات أو خيوط التنفيذ مُتوافرةً للإ ستخدام بين يديه ليستخدمها كما يشاء و متى يشاء مهما اختلف ا سمها أو مكان اعتما دها فى اللغة. إنما الفرق الوحيد بين اعتما د المهمات كمُكوّنٍ لغويٍ مُبا شِرٍ أو كمُكوّنٍ فى المكتبة يظهر مع المُبرمِج البانِى للمُترجِم؛ حيث أنه لو وُضِعت المهمات فى صلب اللغة فسيكون على مُبرمِج المُترجِم وضع ذلك فى حسبانه فى مرحلة المسح scan و التحقق المنطقى semantic analyzing و إنتاج الكود الو سيط. بينما سيكون الامر أخفّ لو كانت المهمات جزءاً من المكتبة البرمجية لا اللغة نفسها؛ فعلي الأقل سيختفي جزءا المسح و التحقق المنطقي و يندمجا ضمن قواعدٍ أخري مبنيةٍ بالفعل.

و سيزدا د الفرق معه لو كان قد قرر بناء الما سِح و المُحقق المنطقى بيديه لا با ستخدام البرامج التى تُنتجهما با ستخدام صِياغات BNF أو EBNF للّغة؛ ففى هذه الحالة يكون زيا دة مُكوّنٍ أو تعبيرٍ برمجيٍ واحدٍ عبئًا على التصميم و البناء لمُكوّنات المُترجِم الأولى التى تعتمد على نَحْو اللغة بشكلٍ تامٍ فى عملها (أعني الما سح scanner و السابر parser).

و نظراً لأننى قد قررتُ بالفعل عدم استخدام الأدوات المُساعِدة فى بناء الواجهة الأمامية لمُترجِم إبداع (علي الأقل في الإصدارات الأولي منه؛ و ذلك للعديد من الأسباب الهامة للغاية) فإننى من هذه النوعية من المُبرمِجين الذين يهمهم بالفعل تقليل مُكوِّنات اللغة إلى الحد الأدنى (بالطبع إذا لم يكن هذا ضاراً) لتخفيف الضغط التصميمى و البِنائى الواقع على كواهلهم.

المُغلِّفات properties:

نبذةٌ بسيطة:

هي نفسها الـproperties في الـC# و التي يُتَرْجِمها الكل بـ(الخصائص) و لكنني أُسَمِّيها بالمُغَلِّفات لسببٍ سيلي ذكره، وهي نوعٌ خاصٌ من الإجراءات؛ حيث أنها تأخذ شكلاً شبيهاً لها في الوظيفة و التعريف، و لكنها تُستَخدَم كما تُستَخدَم المتغيرات: فهي حلقةٌ و سيطةٌ بينهما.

و وظيفتها الأ ساسية هي الإحاطة المنطقية بمتغير أو مجموعةٍ من المتغيرات لتُمَثِّلها أمام العالم الخارجي و تتعامل معه بالنيابة عنها.

و مثالٌ على إ شهارها و ا ستخدامها في الـC# ما يلي:

```
class Data {
   FileStream s;

   public string FileName {
       set {
           s = new FileStream(value, FileMode.Create);
       }
       get {
           return s.Name;
       }
   }
}

Data d = new Data();

d.FileName = "myFile.txt";        // invokes set("myFile.txt")
string s = d.FileName;            // invokes get()
```

أري ألا يُسمَح بها للأ سباب التالية:

يُمكِن با ستخدام الإجراءات فعل نفس الشيء و ذلك كما في المثال التالي بإبداع:

ص = العملاء( )
العملاء(ص + 1)

إجراء (رقم س) العملاء:

```
س = عدد.العملاء


إجراء العملاء(رقم س):
    عدد.العملاء = س
```

و لكن مع الحصول على الفوائد التالية:

- الحفاظ على نَحْو اللغة أبسط و أصغر مما كان سيُصبِح عليه لو تم ضم المُغَلِّفات إلى اللغة.
- القدرة على تحديد مُستوَيات وُصولٍ مُختلِفةٍ لكلٍ من: إجراء التغيير في القيمة، و إجراء الحصول على القيمة.

مثل البرنامج التالي:

```
إجراء عام (رقم س) العملاء:
    س = عددالعملاء
إجراء خاص العملاء(رقم س):
    عددالعملاء = س
```

المُفهْرِسَات indexers:

نبذةٌ بسيطة:

هى مُكوّنٌ أظنه جديداً تماماً تُقدّمه الـC# كطريقةٍ جديدةٍ للتعامل مع الأ صناف، و شرح مايكرو سوفت يقول أن المُفهْرِسات تُمَكّننا من التعامل مع الصِنْف كأنه مصفوفة array. لكنني أرى أن هذا الكلام قد يبدو صحيحاً فى الوَهْلَة الأولى و لكن إمعان النظر فيه يؤدّى إلى الإيمان بخطئه التام، لأن المُفَهْرِسات ما هى إلا نوعٌ من المُغلِّفات properties غير المُسَمّاة و التى تُستَدعى عن طريق ا سم صِنْفها مُبا شرة.

و بالنظر إلى المثال التالي على المُفَهْرِسات سنجد أن التعامل مع الصِنْف كان مُماثِلاً للتعامل مع المُغلّفات لا مع المصفوفات:

```
class MonthlySales {
    int[] product1 = new int[12];
    int[] product2 = new int[12];
    ...
    public int this[int i] {              // set method omitted => read-only
        get { return product1[i-1] + product2[i-1]; }
    }

    public int this[string month] {       // overloaded read-only indexer
        get {
            switch (month) {
                case "Jan": return product1[0] + product2[0];
                case "Feb": return product1[1] + product2[1];
                ...
            }
        }
    }
}


MonthlySales sales = new MonthlySales();
...
Console.WriteLine(sales[1] + sales["Feb"]);
```

و التشابه الوحيد بين التعامل مع المصفوفات و الصِنْف هو ا ستخدام قو سى المصفوفات و هما [ ] فقط لا غير، بينما تُشبه المُفَهْرِساتُ الخصائصَ فيما يلى:

1- بها شِقٌ يُنَفّذ عند قراءة القيمة التى يُعبِر عنها العنصر (قسم get).

2- بها شِقٌ يُنَفّذ عند إسنا د قيمةٍ للعنصر (قسم set).

3- يُمكِن وضع الشِّقين (أيُّهما أو كلاهما) فى الكو د ليَنتج مُفهْرِسٌ قابلٌ: للقراءة و الكتابة، أو للقراءة فقط، أو للكتابة فقط.

إذاً فالتشابه بين المُغلِّفات و المُفَهْرِسات ليس مُجرّد تشابهٍ بل هو تَطابق، و الفرقان الوحيدان: أن المُغلِّفات تُستَدعى باسمها بينما يتم استدعاء المُفَهْرِس باستخدام اسم الكائنِ نفسه، و كذلك السماح بتمرير مُعَامِلاتٍ للمُفَهْرِس للعمل عليها بينما لا يُسمَح بهذا للخصائص العادية، إذاً فالمُفَهْرِسات ما هي إلا مُغلِّفاتٌ غير مُسَمَّاةٍ تقترب من الإجراءات أكثر من المُغلِّفات العادية.

أري ألا يُسمَح بها للأسباب التالية:

ليس للمُفَهْرِسات أي فائدةٍ تتفرد بها و لا يُمكِن القيام بها بشكلٍ آخرٍ ربما يكون بنفس البساطة أو أكثر بساطة، فبإمكاننا إشهار إجراءٍ يَفعل كل ما تَفعل المُفَهْرِسات بدون إتخام اللغة بمُكوّنٍ جديدٍ يزيد من قواعدها بشكلٍ كبير (و هي لا تحتمل من الأصل)، و يُحَمِّل المبرمجين المُتعلّمين لها عبء إدراك الفارق بين المُكوّنات المُختلفة المُتشابهة.

و بصدقٍ: فإن هناك نقطةٌ أخرى تزيدني غيظاً من الـC# هي الإحساس بأن مُصمّميها يُريدون جَعْل الناس يُحِسُّون أن الـC# تُقدّم تقنياتٍ جديدةٍ تَختلف عن كل ما سبق (و خاصة عن الـjava التي تُمثّل لمايكروسوفت نفس ما يُمثّله ابن الضُّرّة للزوجة !) حتى و إن كان مضمون تلك التقنيات يُشكّل مُشكلةً لا حلًا، و عبئاً إضافياً لا تخفيفاً و دعماً !

نموذج الـPython للأصناف و الدوال:

على الرغم من أن لغة الـPython تُمَثِّل بالنسبة لى لغةً من أفضل اللغات (بل أفضلها) من حيث شكل البرنامج المكتوب بها، إلا أنها تحتوى على قراراتٍ تصميميةٍ أجد أنه من الواجب الرد عليها لأنها تُمثِّل قراراتٍ سيئةٍ بما فيه الكفاية لمن كان له قناعاتٌ تصميميةٌ كقناعاتى.

و مِن أَوضَح هذه القرارات الغريبة هي نظرة اللغة إلى الأصناف و الدوال؛ حيث أن هذه النظرة الغريبة تَرى فى الأصناف ما يُشبه الدوال و ترى فى الدوال ما يُشبه الأصناف. فنرى أن الأصناف فى الـPython عبارة عن حاويةٍ لمجموعةٍ من الأوامر المُتَتَابِعة التى تُنَفَّذ بشكلٍ مُتَتَالٍ واحداً تِلو الآخر، و أن المُكوّنات الداخلية للأصناف مثل الدوال و المُشَيِّدات و خلافهما ما هى إلا عباراتٌ تنفيذيةٌ بدورها، و يعنى هذا أنه يُمكِننى أن أقوم بكتابة عبارات إسناد قيمٍ و استدعاءاتٍ لإجراءاتٍ داخل صلب الصِنْف و ليس داخل مُشيّداته أو دواله فقط.

و مثالٌ على هذا:

```
Class class1:
	def func1():
		# code of func1
	A = func1() + 5
	def func2():
		# code of func2
```

حيث يتم فى البداية تعريف دالةٍ تُسمَّى func1، ثم تم استخدامها لإسناد قيمةٍ إلى متغيرٍ يُسمَّى A، ثم تَلَى ذلك تعريف دالةٍ أخرى تسمى func2.

و تكتمل الصورة حينما نَعلم أنه يُمكِننا فى الـPython تعريف دالةٍ داخل دالةٍ أُخرى، و استخدام الدالة الداخلية عن طريق كتابة اسم الدالة الحاوية لها و إتباعها بقوسى دالةٍ نكتب فيهما المُعاملات المُمَرَّرة للدالة الداخلية. و تُعرَف الدوال المُحتوية بالإسم closures.

و مثالٌ على هذا:

```
def  func1(a):
	def  func2(b):
		return a+b
	return a
```

و بالنظر إلى صفات الأ صناف و الدوال في الـPython سنرى أن لكليهما الصفات التالية:

- ا سم الدالة و ا سم أي كائنٍ من الصِنْف هما مُؤَشِّران يُشِيران للدالة أو الكائن، أي أنه يُمكِن التغيير في هوية الدالة أو الكائن التي/الذي يُشير إليه/ـا الا سم.

مثل البرنامج التالي:

```
from math import *

def print_calc(f):
    print "log(%s)=%s, exp(%s)=%s" % (f, log(f), f, exp(f))

print_calc(1.0)
log, exp = exp, log       # evil code! swap the objects the names refer to
print_calc(1.0)
```

حيث أنه في الأمر قبل الأخير جعلنا الا سم (log) يُشِير إلى كو د الدالة (exp)، بينما جعلنا الا سم (exp) يُشِير إلى كو د الدالة (log) !، و النتيجة التالية لا ستدعاء الا سمين قبل التبديل و بعده تدل على أثر التبديل و نجا حه:

```
log(1.0)=0.0, exp(1.0)=2.71828182846
log(1.0)=2.71828182846, exp(1.0)=0.0
```

- كلاهما يُمثِل كتلةً من الأوامر التي تُنَفَّذ بشكلٍ تَابعي.

- كلاهما له القدرة على احتواء تعريفاتٍ لأ صنافٍ دا خله، أي أنه يُمكِن للدالة أن تحتوي على تعريف صِنفٍ من الأ صناف دا خلها، لتَعُود و تستخدمه بصورةٍ طبعيةٍ كما يحدث في حالة الأ صناف المُتفرِّعة في باقي لغات البرمجة.

مثال:

```
def  func1(a):
        Class class1:
                def func2():
                        # code of func1
                A = func2() + 5
                def func3():
                        # code of func2
```

- كلاهما له القدرة على احتواء دوالٍ داخليةٍ يتم استدعاؤها عند الحاجة إليها عن طريق اسم المُكوّن الحاوى لها، مع الفارق فى أن الدالة تحتوى داخلها دالةً داخليةً واحدةً فقط بينما للصِّنْف تعديد الدوال. و كذلك فإن الدوال التى فى الصِنف يتم استدعاؤها باسمها بينما تُستَدعى الدالة الداخلية باسم المُتغير الذى استَدعَى دالتها الحاوية.

مثال 1:

```
def  func1(a):
        def  func2(b):
                return  a+b
        return  a

a = func1(5)
b = a(10)
```

مثال 2:

```
Class class1:
        def func1():
                # code of func1
        A = func1() + 5
        def func2():
                # code of func2

Class1  inst
inst. func1()
```

و من الصفات السابقة نرى أن هناك تقريباً لا معنى له بين صفات مُكوّنين برمجيين يجب إبقاء كلٍ منهما مُنفصلاً عن الآخر، و عدم إثارة البلبلة التى لا داعى لها بجعل كلٍ منهما يَحمل شيئاً من صفات الآخر بشكل لا يُحقِق أية أهدافٍ فى حقيقة الأمر.

فالدوال هى فى الأصل كتلةٌ من الأوامر التى تُنَفّذ بشكل تَتابعى بحيث يتم استدعاؤها عند الحاجة إليها؛ فيُوفّر وُجودُها تكرارَ الأكواد و مساحةَ البرنامج النصية، و يزيد القدرة على اكتشاف الأخطاء و السرعة فى ذلك و غيرهن من العوامل الأخرى، أما الأصناف ففى الأصل لم تَنشأ إلا لتكون حاويةً تجمع بين مجموعةٍ من المُتغيرات و مجموعةٍ من الدوال التى تعمل عليها، و عن طريق الإستنساخ من الأصل يتم الإستفادة

من كود الصنف أقوى استفادةٍ عن طريق عزل المُحتَويات الداخلية للكائنات عن المُبرمِج النهائى و غيرها من الفوائد.

فما معنى وجود دالةٍ تَحوى دالةً أخرى داخلها، أو صِنْفٍ يَحوى أوامراً تُنَفّذ بشكلٍ مُتسلسِلٍ و كأنها كتلة أوامرٍ فى صلب دالةٍ من الدوال ؟!

و ما الفائدة التى ستعود علينا من ذلك المزج و التقريب الغريبين ؟،

و أى ضررٍ سيعود علينا عند التفريق بينهما كما تفعل بقية اللغات ؟

الدّامِجَات unions:

نبذةٌ بسيطة:

الـunions فى الـC و الـ++C و ما حذا حذوهما من اللغات: هى مُكوّناتٌ الفكرة الرئيسة من ورائها هي إتاحة الفرصة لاستخدام نفس الحَيّز من الذاكرة بأي نوع من الأنواع المختلفة. يعنى أن الدّامِج عبارة عن حاويةٍ تُشبِه الصِنف من حيث أنه يتم داخله تعريف مُتغيرات، و لكنه يختلف عنه فى أن الصِنف يَعتبر كل مُتغيرٍ داخلي مُنفصِلاً عن المُتغيرات الأخرى من حيث حَيّز الذاكرة المحجوز له. بينما فى الدّامِج يتم إعطاء كل تلك المتغيرات مساحةً مُشترَكَةً فى الذاكرة تبدأ من نفس النقطة و تنتهى حسب مساحة أكبر المتغيرات حجماً، و لكن لا يُمكِن لأي متغيرٍ أن يتعامل إلا مع جزءٍ من تلك الذاكرة المُشترَكة يتناسب مع نوع ذلك المتغير.

يعنى لو أننا كتبنا ما يلى:

```
// declaring_a_union.cpp
union DATATYPE    // Declare union type
{
    char   ch;
    int    i;
    long   l;
    float  f;
    double d;
} var1;          // Optional declaration of union variable

int main()
{
}
```

فإن معنى هذا أننا نُعَرِّف متغيراً يسمى var1 من النوع DATATYPE يُمكِننا التعامل معه على أساس:

أنه من النوع char بالإسم ch.

أو النوع int بالإسم i.

أو النوع long بالإسم l.

أو النوع float بالإسم f.

أو النوع double بالإسم d.

و القيمة التى نتعامل معها كل مرةٍ هى نفس القيمة و لكن مع اختلاف المساحة التى نتعامل معها منها.

أري ألا يُسمَح بها للأسباب التالية:

أري أن الأسباب التى يَسُوقها الشارحون للبرمجة بلغتى الـC و الـ++C فى كتبهم لوجود الدّامِجَات و مدى فائدتها لا تُسمِن و لا تُغنِي من جوعٍ أمام التحقيق و التدقيق؛ فما وجدتُ منها غير:

- أن الدَّامِجَات تُوَفِّر فى ا ستخدام الذاكرة حينما نريد التعامل مع قيمةٍ ما بأكثر من أ سلوب، فبدلاً من: تعريف مُتغيراتٍ مُختلفة الأنوا ع بشكلٍ مُنفصل، و أقوم كل مَرّةٍ حينما أحتاج إلى التعامل مع القيمة على أنها من نوع معين بوضعها فى المُتغير الذى ينتمى إلى ذلك النوع، و من ثم أقوم بإجراء العمليات على ذلك المتغير، أقول: بدلاً من كل هذا فإننى أقوم مُبا شرةً بوضع تعريفٍ لدامج يدمج بين تلك الأنوا ع فى مكانٍ وا حدٍ لأتمكن من التعامل مع القيمة المُخَزّنة فى دا خله بأى شكلٍ أريده.

و هذا سببٌ لا أظنه موجوداً اليوم؛ فهذا القول كان صحيحاً حينما كانت الأجهزة التى تتم البرمجة لها لا تحتوى من الذاكرة إلا على 64 كيلوبايت أو ما يُشابهها. يَستَغِل نظام التشغيل القليل من الكيلوبايتاتٍ منها و تَستَغِل البرامج العا دية الباقى، و هذه مُوا صفاتٌ قريبةٌ من مُوا صفات أجهزة الـPDP11 التى كانت أوائل برامج الـC مكتوبةً لها، و ما عا صَرَها من الأجهزة.
و لكننا الآن نعيش فى زمنٍ مُختلفٍ ليس من المُمكِن فيه أن تُوهِمنى أنك حينما تكتب برنامجاً كالسابق سوف تُوَفِّر عليّ الكثير من الذاكرة المُهدَرة، بل العكس أنك ستحصل على برنامجٍ غريبٍ سئ التكوين مُقابِل الحصول على بضعة بايتاتٍ لا تساوى أى شئٍ بالمُقارَنة مع أقل الإمكانيات فى سعات ذواكر الحا سوب الموجو دة اليوم.

إ ذاً فهذه الفائدة التى كانت موجو دة للدامجات قد انتهت إلى غير رجعةٍ و كان من المُحَتّم على مُصمّمى لغات البرمجة التنبه إلى هذا و إبعا د ذلك المُكوّن الزائد عن اللغات الجديدة، لا ضـمّه إليها كما حدث مع الـC++ التى سارعت إلى ضمه إليها و كأنه كنزٌ ثمين.

- الدّامِجات أكثر كفاءةً من ا ستدعاء الإجراءات التى تُحَوّل بين أنوا ع المتغيرات، أى أنه فى كل مرةٍ نحتاج فيها إلى تحويل قيمةٍ ما من نوعٍ إلى نوعٍ آخر: فبدلاً من ا ستدعاء إجراءاتٍ للتحويل بين أنوا ع المُتغيرات فإننا نستخدم المتغيرات الموجو دة فى الدّامِج مبا شرة، و نستخدم منها ذلك الذى له النوع الذى نرغب فى التعامل مع القيمة على أ سا سه.

و هذا قولٌ آخرٌ أراه غير منطقي في عالم اليوم؛ حيث أن مثل هذه التوفيرات الغريبة لا تُقـدِّم تلك النتائج الخارقة التى تُغرِى ببذل الجهد و التضحية للحصول عليها،
فما الذى سيضر البرنامج إ ذا ما ا ستدعى إجراء تحويلٍ عشر مراتٍ أو أكثر ؟!،
و ما التأخير الذى سيحدث عندئذ ؟!،
و كيف سيُؤثِّر ذلك التأخير على زمن تنفيذ البرنامج بصورةٍ و لو طفيفة ؟!

أين الإبداع فيها ؟

هذا الجزء مُخصَّصٌ لشرح الأسباب التي جعلتني أُصمِّم إبداع بالشكل الذي ظَهرتْ عليه في النهايـة، فرغـم أن هناك شرحاً لقواعد التصميم التي أقتنع بها في فصلٍ سابقٍ، و أن هنـاك شـرحاً للمُكوِّنـات الـتي رَفضـتُ ضمَّها لإبداع و الأسباب التي دفعتني إلي ذلك: إلا أن الحاجة لهذا الجزء ماسَّةٌ للغاية؛ و ذلك لأنـه حـتي عند ذِكْر الأصول الفكرية لأي منهجٍ من المناهج فسيحدثُ اختلافٌ في تفسير تلك الأصول و تنزيلهـا علـي الواقع، و الأمر الوحيد الذي سيمنعُ مثل ذلك التشتت التفسيري هو ضـرب الأمثلـة العمليـة لـذلك المنهـج حينما يتم تنزيله علي الواقع العملي بواسطة من وَضَع تلك الأصول.

و قد التزمْتُ هنا قَدْر الإمكان بالبساطةِ الشديدة حتي لا يتضخم حجم الكتاب بشـكلٍ لا يُطاق، و الإكثارِ مـن الأمثلة العملية حتي تُصبح الأمور أكثر سهولةً من حيث الاستيعاب. و كـذا فقـد أدرجـتُ أمـوراً ربمـا يـري القاريء أنها ليست بالأهمية التي يجب معها الإهتمام بها و إفراد مكانٍ لها، و لكني فعلتُ هذا كتوضيح لمبـدأ أن لغة إبداع ليست مُجرد تجميعةٍ من المُكوِّنات وُضِعت مع بعضها البعض لتصبح لغـة برمجـةٍ عاديـة، إنمـا هي لغةٌ صُمِّمت بمُنتهي العناية حتي تتوافق كل مُكوِّناتها مع بعضها البعض و تنسجم في تناغُم جميل.
و ربما لا تظهر الحكمة من وراء جعل جزءٍ ما من اللغـة علـي الشـكل الـذي هـو عليـه بشـكلٍ مُباشِـر، و لكـن معرفة الكثير من التفاصيل الأخري تجعل الذهن يتنبه إلي ما فاته و يُدرِك التوليفة التي جعلت الأمور تُصبح هكذا.

أسباب جعل لغة إبداع لغةَ تفسيريةَ بجانب وجود مُترجِمٍ لها:

- للإستفادة من فلسفة الـjava التي تَنُص على كتابة الكود مَرَّةً واحدةً و تشغيله فى أي مكان، و بالتالى: نُنتِج البرنامج و نُترجِمه مرةً واحدة، و نَرفعُ نسخةً واحدةً على الشبكة، و يُحَمِّل المُستخدِم النهائى نسخة وحيدة، إذا رغبنا في هذا.

- تُمَكّننا من نقل البرنامج بين أنظمة التشغيل بحريةٍ و ذلك بدون أن يُضارَّ المُبرمِج فى أى شيء؛ إذ أنه يتعامل مع البيئة التشغيلية التى سيتم إعطاءها القدرة علي أنظمة تشغيلٍ أخري، و كذلك تحديثها للتعامل مع الإصدارات الجديدة من أنظمة التشغيل التي تم تحديثها، و هذا يَعنِى حياةً أطول للبرامج و محمولية عاليةً جداً بدون أدني تعبٍ من جانب المُبرمِج.

- إذا ما أردنا بعد ذلك تطوير لغة برمجةٍ عربيةٍ أخرى مُتخصِّصةٍ في مجالٍ مُعيّنٍ (كلغةٍ للتعبيرات و التعاملات الإحصائية علي سبيل المِثال) فإن إنتاجها سيكون أسهل؛ و ذلك لأننا لن نحتاج حينئذٍ إلا إلى صُنْع الواجهة الأمامية من مُترجِم تلك اللغة و مُحَوّلٍ إلى الصيغة الوسطى الخاصة بالآلة الوهمية لإبداع فقط، دون تَجَشُّم عناء صُنْع مُترجِمٍ كامِل.

- هذا هو التطور الطبعى فى علوم الحاسب البرمجية؛ التى ظلت رُوَيْداً رُوَيْداً تُبعِد البرامج عن التعامل المباشر مع العَتاد الصلب و تُسنِد هذا الدور إلى النُواة kernel، و لم يبق فى الوقت الحالى إلا إبعاد البرامج عن التعامل المُباشِر مع المُعالِج و هو الأمر الذى يتحقق باستخدام المُفسِّرات. و هكذا نكون فى مَأمَنٍ من أغلاط المُبرمِجين فى التعامل مع المُعالِج فى التطبيقات التى لا يهمهم فيها كيفية تنفيذ أوامرهم فى المُعالِج ما دامت تُنَفَّذ بكفاءةٍ و قوة.

- في حالة استخدام بيئات التشغيل: فإن التطورات التقنية الجديدة لن تكون مُطَبَّقةً فقط على البرامج الجديدة كما فى حالة الترجمة منذ البداية، بل ستنطبق بأثرٍ رجعيٍ على البرامج القديمة التى تعمل على الإصدارات الحديثة من بيئة التشغيل.

فوائد وجود نظام أنواعٍ ثابت static typing:

- إكتشاف أخطاء استخدام الحَاوِيات (أي المُتغيّرات و الثوابت و كائنات الاصناف و كائنات الألقاب) في زمن الترجمة compile time بدلاً من زمن التنفيذ run time.
- تخفيف العبء عن زمن التنفيذ؛ و ذلك لأن إختبارات الأنواع و صحة استخدامها تتم في هذه الحالة في زمن الترجمة مرةً واحدة، بينما في حالة نظام الأنواع المُتغيّر dynamic typing تتم الإختبارات في زمن التنفيذ في كل مرةٍ يتم فيها استخدام الحاوِية.

- إمكانية ا ستغلال معلومات الأنواع (المُتوافِرة في زمن الترجمة) للمزيد من تحسين ا ستغلال الذاكِرة و/أو المُعالِج و أخذ ذلك في الحسبان أثناء زمن الترجمة.

فائدة وُجوب تعريف الحاويات يدوياً :

- يُعَد نوعاً من التوثيق الواضِح التِلقائي.
- يُكسِب الكو د مقروئية أعلي.
- تخفيف العبء من علي كاهل المُترجِم و مُبرمِجيه؛ حيث أنه أبْعَد عنه عبء ا ستنتاج الأنواع المُختلِفة بنفسه، مع العلم أنه لم يحمل عنه عبء ا ستنتاج مساحات الحاويات المُختلفة التي يُمكِن أن تمتلك مِساحة مُتغيّرة حسب ظروف الإستعمال (مثل المُتغيّرات الرقمية).
- في كثيرٍ من الأحيان في اللغات التي تُتِيح للمُبرمِج ا ستخدام المُتغيّرات بدون تعريفها يدوياً سنُضطَّر لمُحاكاة عملية التعريف، مثل الموقف الافتراضي التالي:

```
س = {} \ لتعريف جدولٍ فارغٍ لا ستخدامه داخل تعبير بينما /
بينما < شرط.منطقي> :
    <كو د.بينما>
```

حيث احتجنا في هذه الحالة إلي تعريف جدولٍ قبل الدخول إلي صلب تعبير بينما؛ لأننا لو قُمنَا بتعريفه داخل التعبير فسيتم اعتباره جدولاً مُختلِفاً في كل لفة ! و بالتالي يَأخذ قيماً غير التي نتوقع له أن يأخذها.

و لذلك نُضطَّر إلي إسنا د قيمة جدولٍ فارغٍ إلي الإسم (س) حتي يَعلم مُترجِم اللغة أن (س) ما هو إلا جدولٌ نَرغب في تعريفه. و مثل هذا المواقف يُمكِن رؤيته في لغاتٍ مثل python و ruby و غيرهن من لغات التنويع المُتغيّر.

- يُمكِن جَعْل التعريف و الإسناد في سطرٍ واحدٍ كمُحاكاةٍ للّغات الديناميكية و بالتالي لن يأخذ التعريف أسطراً زائدة و لن يَجعل الكود أطْوَل، كما يُمكِن جَعْل التعريفات في أعلي الأوامر التنفيذية كتوثيق واضح كما يحدث إجباراً في لغة object pascal.
- في اللغات التي لا يتمٍ فيها تعريف الحاويات داخل الأصناف في أسطرٍ مُنفردة (مثل لغتَيْ الـ python و ruby) فإنه لمعرفة الحاويات داخل صِنفٍ ما: ستُضطَّر لقراءة أكواد إجراءاته كلها، أو ستعتمدُ علي برنامج يُساعِدك علي هذا، أو ستحتاج لتوثيقاتٍ خارجية تُخبِرك عَمَّا تُريد. أما في اللغات التي تُجبِرك علي التعريف اليدوي فلن تحتاج لفعل أيٍ من هذا لأن الكود في أعلي الصِنف يُغنِيك عنه.

فوائد عدم تعبير الأنواع البسيطة عن الأحجام :

- يُؤدِّي إلي قِلّة عدد الكلمات المفتاحية في اللغة. فالأنواع int و uint و float و double و short و decimal و غيرهن الكثير تم اختصارها في النوع رقم. بينما تم اختصار الأنواع String و char في النوع نص.
- عدم التقيد بمَفاهِيم الذاكرة الحالية (مثل الإعتماد علي عدد البايتات في تحديد المِساحة) و بالتالي الإنفصال التام عن تكوين الآلة التي يعمل علبها البرنامج. إلا عند استخدام أكواد التجميع التي ستتغير إذا تغيرت البِنية العَتادية بما لا يُؤثِّر علي نَحْو اللغة نفسها. و كذا بما لا يُؤثِّر علي توحيد معني كل قاعدةٍ في اللغة مهما اختلف نظام التشغيل أو المُعالِج اللذَيْن تعمل عليهما برامجها.
- تقليل عدد القواعد في اللغة من حيث:
  - عدم الحاجة لقواعد التخصيص casting.
  - عدم الحاجة لوجود الأكواد العامة generic codes بكل ما تحتويه من صُعوباتٍ و تعقيداتٍ جَمّة. فالتعميم موجودٌ عملياً في الأكواد؛ فليس هناك تفريقٌ بين الأحجام و لم يُجعَل كلُّ حجم نوعا مُختلفاً عن الآخَر.
- التناغم مع التفكير البَشري الطبعي عالي المُستوَي الذي لا يَهتم إلي حدٍ كبيرٍ بالتحديد الفائق للأمور، و بالتالي زيادة سرعة التعلُّم للُّغة. مع ترك أمر هذه التحديدات للمُترجِم أو المُفسِّر لكي يهتم هو بها حسب حالة الآلة التي يعمل عليها و ظروف العمل.
- زيادة المقروئية.
- زيادة المكتوبية.

سبب جعل تعريفات الألقاب و الإجراءات و الأصناف في نهاية الملفات الرئيسة. و بعد الاستخدام الفعلي لها:

- لو كان العكس هو ما حدث (أي تعريف تلك المُكوّنات قبل استخدامها): فسيكون مُؤلِماً للمُبرمِج في كل مَرّةٍ يفتح فيها برنامجاً أن يُقلِّب الكود بحثاً عن بداية أوامر البرنامج التي تُوجَد أسفل التعريفات. لكن في حالة إبداع فبمُجرَّد فتْح الملف الرئيس يَجِد المُبرمِجُ بدايةَ الأوامر دون جهدٍ منه.

و في لغاتٍ كالـpython يلجأ المُبرمِجون لعمل إجراءٍ له في العادة الإسم main ثم يستدعونه في بداية الأوامر بالأسفل. و في داخل هذا الإجراء يضعون أوامر المسار الرئيس للبرنامج كاملةً كما هو الحال مثلاً في لغة الـC. أي أنهم يُحاكُون أسلوب اللغات الوظائفية و الإجرائية للتغلب علي تلك المُشكِلة.

مثال:

```
def main :
```

```
        # main program's code

def func1 :
        # function1's code
        .
        .
        .
main()
```

بينما حَلَلْناها بشكلٍ أبسط و لا يتعارض مع بقية الأسباب التالية.

- هذا يتفق مع الطريقة التلقائية التي يَعمل بها المُبرمِج؛ حيث عندما يقوم المُبرمِج بكتابة خُوارِزْمٍ لحل مُشكِلةٍ ما فإنه يقوم بـ:
    - كتابة الخطوط العريضة العامَّة، و يَستخدِم فيها " صناديق سوداء" black boxes عبارة عن مُكوّناتٍ موجودة، أو سيقوم بإنشائها بالكامل فيما بعد.
    - فَكُّ الخطوط العريضة في المرحلة السابقة و توضيح كيفية عَمَل كل صندوقٍ أسودٍ تم استخدامه من قبل.

و هذا ما يحدث في إبداع، حيث :

- يَقوم المُبرمِج بكتابة المسار العام للبرنامج باستخدام إجراءاتٍ وهميةٍ لم تُكتَب بعد؛ لأداء المهام الكبيرة و/أو الصعبة و/أو المُتكررة، و أصنافٍ وهميةٍ لتُمثِّل دور هياكِل البيانات التي يحتاج إليها في عمله، و كذلك ألقابٍ وهمية.
- يَقوم المُبرمِج بكتابة أكواد تلك الإجراءات و/أو الأصناف و/أو الألقاب في المرحلة التالية بحيث يكتمل الكود تماماً.

و نُلاحِظ هنا التماثل التام بين الأسلوبين و الذي يجعل طريقة إبداع مُناظِرةً للعمل التلقائي البشري، و هذا (من جديد) يجعل اللغة أسهل في التعلّم للأطفال الصِّغار و المُبتدئين.

أسباب إمكانية جعل تعريفات الألقاب و الإجراءات و الأصناف مختلطة، و عدم إجبار المُبرمِج على تعريف الألقاب سوياً و الإجراءات سوياً و الأصناف سوياً:

- تسهيلاً على المُبرمِج؛ ففي حالة وُجُوب ضَمّ المُتشابهات فسيكون من المُرهِق عند تعريف مُكوّنٍ جديدٍ البحثُ عن مكان الفِئة التي ينتمي إليها ذلك المُكوّن لكتابة تعريفه هناك، و في كل مَرَّةٍ نُعرِّف فيها إجراءاً جديداً سنضطر للبحث عن مكان تعريف الإجراءات لكتابة ذلك الجديد هناك، و كذا مع الأصناف و الألقاب !
- زيادةٌ غير مُبرَّرةٍ للقواعد لأنها لا تأتي بفائدةٍ مُعيّنةٍ لا يُمكِن الاستغناء عنها.
- لإعطاء المُبرمِج القُدرة على ضَمّ المُكوّنات مُختلِفة الأنواع و التي ترتبط وظائفياً مع بعضها البعض، بحيث عند التعديل في كيفية القيام بتلك الوظائف لا تكون هناك حاجةٌ للذهاب بعيداً، و كذا عند

الحاجة لتطبيق أمرٍ من الأوامر علي المُكوّنات المُختلِفة التي تُؤدّي وظيفةً مُعيّنة (مثل تحويلهن إلي تعليقٍ لسببٍ من الأسباب).

أسباب وُجوب تعريف أي صنفٍ خارج الملف الرئيس في ملف باقةٍ مُستقلٍ له نفس اسمه:

- في البرامج الصغيرة لن يحتاج المُبرمِج إلي ملفات باقاتٍ علي الإطلاق، لأنه سيضع الأصناف التي يحتاجها في الملف الرئيس لصِغَر حجم الكود كله، و بالتالي لن يُشكّل هذا الأمر له أي فارق.
- في البرامج المُتوسِّطة و الكبيرة سيكون من الأفضل جعل كل صِنْفٍ في ملفٍ مُستقلٍ لما يلي:
  - ضخامة الأصناف في البرامج الكبيرة تجعل من الأفضل توزيع كل صِنْفٍ في ملفٍ مُنفصلٍ لضمان صِغر حجم الأكواد في كل ملفٍ قدر الإمكان.
  - سرعة الوصول إلي الصنف المرغوب في الوصول إليه.
  - المُساعَدة القُصوي للمُترجِم في أداء عمله بسرعةٍ و بساطة (و هذا يرتبط ببناء المُترجِم الداخلي).
  - المُساعَدة علي إعادة الإستعمال القُصوَي بمُجرَّد نسخ الملف و لصقه في المشروع الجديد.
  - تَماشِياً مع مبدأ "تشابه الإفتراضي مع المَلْمُوس" حيث في إبداع تُوجَد القاعدة :"الملف = صنف، و المُجلّد = باقة" و التي تُسهّل التعلُّم علي المُبتدئين و صِغار السن، لأن هؤلاء لن يكون بإمكانهم إستيعاب المفاهيم المُجرّدة (مثل: الباقات) إن لم ترتبط في أذهانهم بشيءٍ ملموسٍ يرونه بأعينهم (مثل: المُجلّدات).

أسباب جعل تعبير (من – إلي – بقيمة -) بهذا الشكل:

- في هذا التعبير فإن قيمة الزيادة (التي تُوضَع بعد كلمة بقيمة) تُجمَع علي الحَدِّ السابق في المُتسلسِلة series لإنتاج الحَدِّ الجديد، و سبب ذلك:
  - هذه هي الطريقة الوحيدة لإنتاج المُتَسَلْسِلات في الأحيان التي لا يكون هناك اسمٌ للحَدِّ السابق نعتمد عليه عند وضع تعبير قيمة الزيادة (مثل حالة: بينما – في من – إلي – بقيمة -).

حيث في تعبير (بينما) يُمكن وَضْع:

```
بينما أ في من 1 إلي 100 بقيمة أ+2 :
    \ الكود هنا /
```

لكن في حالةٍ مثل :

```
رقم{1} الأرقام = {من 2 إلي 6 بقيمة 2}
```

لا يكون هناك رمزٌ للحَد القديم لوضعه في مُعادَلة حساب قيمة الحَد الجديد، و بالتالي يجب أن يتم هذا الأمر ضِمنياً.

- يُمكن مُحاكاة الحالة التي لا يكون فيها الحَد الجديد خاضعاً لصيغة:

الحد الجديد = الحد القديم + قيمة انتقالية

و ذلك عن طريق جعل القيمة الإنتقالية كما يلي:

القيمة الانتقالية = - الحد القديم + قيمة إنتقالية داخلية

و بالتالي تكون مُعادَلة حساب الحَد الجديد قد تَحوّلت لما يلي:

الحد الجديد = الحد القديم + -الحد القديم + قيمة انتقالية داخلية

الحد الجديد = قيمة انتقالية داخلية

مثال: البرنامج التالي:

```
رقم الرقم
رقم{3} الأرقام = {                                  |
{ {من 1 إلي 31 بقيمة قيمة()} {1  2  3} {1  2  3} }  |
{ {من 2 إلي 200 بقيمة 2} {1  2  3} }    }
بينما الرقم في الأرقام :
    أكتب.نص. سطر("قيمة حد الأرقام الجديد هي " + إلي.نص(الرقم))

إجراء (رقم الحد.الجديد) قيمة :
    رقم مشترك الحد.القديم = 1
    الحد.الجديد = - الحد.القديم + 2 × الحد.القديم + 1
    الحد.القديم = 2 × الحد.القديم + 1
```

له الخَرْج:

```
قيمة حد الأرقام الجديد هي 1.0
قيمة حد الأرقام الجديد هي 3.0
قيمة حد الأرقام الجديد هي 7.0
قيمة حد الأرقام الجديد هي 15.0
قيمة حد الأرقام الجديد هي 31.0
قيمة حد الأرقام الجديد هي 1.0
```

.
.
قيمة حد الأرقام الجديد هي 200.0
قيمة حد الأرقام الجديد هي 1.0
قيمة حد الأرقام الجديد هي 2.0
قيمة حد الأرقام الجديد هي 3.0

فهنا حينما أردنا عمل صَفٍ له القيمة التالية { 1 3 7 15 31} أي أن :

الحد الجديد = الحد القديم × 2 + 1

كتبنا الكود:

رقم{1} الأرقام = { من 1 إلي 31 بقيمة قيمة() }

إجراء (رقم الحد.الجديد) قيمة.انتقالية :
رقم مشترك الحد.القديم = 1
الحد.الجديد = -الحد.القديم + 2 × الحد.القديم + 1
الحد.القديم = 2 × الحد.القديم + 1

أسباب جعل تعبير (من – إلي – بقيمة -) محصوراً علي القيم الرقمية فقط دون كائنات الأصناف و كائنات الألقاب:

- التعبير لا يُستخدَم فقط في تعبيري بينما و لو، بل يُستخدَم كذلك في إعطاء الجداول قيمة. مثال:

رقم{1} الأرقام = {من 10 إلي 100 بقيمة 10}

و بالتالي مَدُّ قدرات التعبير لتشمل كائنات الأ صناف و الألقاب سيجعل ا ستخدامه في إعطاء قيم لجداول كائنات الأ صناف و الألقاب صعباً جداً و له قواعد مُعقَّدة. لذا كان الهرب من كل هذا.

- عدم القدرة علي ا ستخدام هذا التعبير مع كائنات الأ صناف و الألقاب لا يعني عدم القدرة علي أ داء نفس العمليات عليهما. و المثال التالي يوضح هذا:

الكائن = أول.كائن
بينما الكائن # آخر.كائن :
\ افعل شيئاً ما /
الكائن = الكائن_التالي

و لو كان بالإمكان كتابة هذا بتعبير (من – إلي -) لكان كالتالي:

```
بينما الكائن في من أول.كائن إلي آخر.كائن بقيمة الكائن = الكائن_التالي :
    \ افعل شيئاً ما /
```

و الفرق بين الكودين هو سطران لا أكثر، و في حالة كون التعبيرات التي تلي كلمات من و إلي و بقيمة مُعقَّدةً فربما نحتاج إلي جعل كل واحدٍ منها في سطرٍ مُستقِل، مثل الكود التالي:

```
بينما الكائن في من تعبير.الحصول.علي.القيمة.الإبتدائية      |
               إلي تعبير.الحصول.علي.القيمة.النهائية        |
               بقيمة  تعبير.القيمة.الانتقالية :
    \ افعل شيئاً ما /
```

و نظيره المُكافِيء له با ستخدام الطريقة العا دية هو:

```
الكائن  =  تعبير.الحصول.علي.القيمة.الإبتدائية
بينما الكائن # تعبير.الحصول.علي.القيمة.النهائية:
    \ افعل شيئاً ما /
    تعبير.القيمة.الانتقالية
```

فنَرى هنا أن الفرق تلا شي، مع الإحتفاظ لكلا التعبيرين بالوُضوح و البُعْد عن التعقيد.

سبب عدم ضم ما يُماثِل المُعامِل المنطقي XOR كمُعامِلٍ قياسي في اللغة:

لأن المُعامِل # يقوم بذات الوظيفة !، و انظروا إلي جدولي الصواب لكلٍ منهما لتروا أن الأمر صحيح:

جدول صواب المُعامِل #

صحيح	صحيح	خطأ
صحيح	خطأ	صحيح
خطأ	خطأ	خطأ
خطأ	صحيح	صحيح

جدول صواب المُعامِل XOR

صحيح	صحيح	خطأ
صحيح	خطأ	صحيح

خطأ	خطأ	خطأ
خطأ	صحيح	صحيح

و هما مُتماثِلان كما هو واضح.

لكن الأمر الغريب و المُثير للسخرية أن لغات برمجةٍ شهيرةٍ تُقدِّم المُعامِل XOR مع أنه يمكِنها الاستغناء عنه كما فعلنا في إبداع كتجلٍ جديدٍ من تجليات عدم محاولة جعل تلك اللغات تتبع نظام " حِمْيَةٍ " منا سب !

يُمكِن كتابة :

لو س في { 1 2 3 4 5 6 7}:
\ أوامر /

و لا يُمكِن كتابة :

لو س في { {1 2 3} {4 5 6} }:
\ أوامر /

و ذلك لأنه من غير المنطقي كتابة الصيغة الثانية بينما الأولي تؤدي الغرض و لا تُرهِق مُبرمِج المُترجِم. كما أن إجبار المُبرمِج علي جعل كوده بالصيغة الأولي و ليس بالصيغة الثانية يجعل الكود أكثر مَقروئية. خذ الجدول الذي في المثال التالي حتي تُصبح الصورة أوضح:

لو س في { {{1 2 3} {4 5 6} {7 8 9}} {{10 11 12} {13 14 15} {16 17 18}}}:
\ أوامر /

و الآن قارِن بينه و بين المثال التالي:

لو س في { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 }:
\ أوامر /

هل رأيتَ الفرق ؟

سبب الإبقاء علي مُعامِلات الوصول و الاستنساخ في المسار العام للبرنامج و الإجراءات رغم أنها لن تشكل فارقاً:

- عدم إرهاق المُبرمِج بمعرفة أنه :
  - لا يَصِح استخدام مُعامِلات الوصول في المسار العام أو الإجراءات، بل يَصِح في الأصناف فقط.
  - لا يَصِح استخدام الاستنساخ في المسار العام، بل يَصِح في الإجراءات و الأصناف فقط.
  - الثبات يُمكِن استخدامه في أي مكان.

  أما في حالة الإبقاء على إمكانية كتابة الصفات التي يُريدها المُبرمِج في أي مكانٍ باختلاف المعاني حسب السياق فسيكون عليه أن يُدرِك تلك المعاني (البَدَهِية البسيطة) فقط، و ليس عليه التنبه لما يَصِح و ما لا يَصِح.
- عدم إرهاق مُبرمِج السَّابِر parser بحالاتٍ و قواعدٍ ليس لها داعٍ بينما يُمكِن تسهيل الأمر عليه و المُستخدِم في ذات الوقت.

سبب جعل إعطاء القيم الافتراضية لمُعامِلات الإجراءات عن طريق ترتيبها، و ليس بالإسم (named parameters) :

- لو تم عمل مكتبةٍ استُخدِمت فيها طريقة الـnamed parameters ثم استخدمَها الناس بالفعل، ثم قُمْنا بتغيير أسماء المُعامِلات (أو حتي واحدٍ منها فقط): فسوف يحدث انهيارٌ لأكواد المُبرمِجين التي تَستخدِم الإجراءات المُعتمِدة علي هذه الطريقة في المكتبة !
- تسهيلًا علي مُبرمِج السَّابِر و المُبرمِج مُستخدِم اللغة؛ لأنه حتي لو دَعَمْنا الطريقة الثانية فسنكون مُضطَّرين لدعم الطريقة الأولي !، أي أن الطريقة الثانية لا تُمثِّل إلا زيادةً في القواعد يُمكِن الاستغناء عنها، فلِمَ لا نستغني عنها إذاً ؟!
- وجود الطريقة الأولي فقط أسهل في الشرح و التعلّم من شرح الطريقتين الأولي و الثانية.

سبب الاستغناء عن جملة do – while:

الحق أنه كان قراراً سهلاً جداً لأن التعبير التالي يُؤدِّي نفس الوظيفة:

بينما صحيح :
\ أوامر/
لو \ شرط / :
أنهي()

و ما دام هناك البديل السهل الذي لا يُؤدِّي إلي زيادة عدد القواعد: فلِمَ نُدرِج قاعدةً أخري إضافية إلي اللغة ؟!

من الأسباب التي تجعل من الأفضل عدم ضم المُعامِلات المنطقية الحسّاسة مثل | & في لغة الجافا :

- تُؤدّي إلي الوُقوع في أخطاءٍ منطقيةٍ قد لا يَنتبه المُبرمِج إلي سببها بسهولة؛ نتيجةً لعدم تنفيذ المُعامِل للشِّق الثاني من العملية المنطقية و الذي قد يكون استدعاءاً لإجراءٍ له أعراضٌ جانبية side effects. و بالتالي لا يتم استدعاء الإجراء و هو ما يتسبب في عدم تنفيذ الأعراض الجانبية مع أن المُبرمِج قد اعتمد علي أنها ستُنفَّذ !

و هذا واحدٌ من أعوص المشاكل في الإكتشاف؛ نظراً لأن عين المُبرمِج تمر مرور الكِرام علي موضع الغلط دون الإنتباه إليه (و أقول هذا عن تجربةٍ واقعية).

- يُمكِن مُحاكاة هذا السلوك التحسيني بوضع القواعد التالية في المُترجِم و المُفسِّر:
  - إذا كان في العملية استدعاءٌ لإجراءٍ فسيتم تنفيذه لا محالة؛ لتجنب افتقاد أعراضه الجانبية.
  - إذا لم يكن هناك استدعاءاتٌ لإجراءاتٍ فسنُطبِّق طريقة المُعامِلات الحَسّاسة. و بالتالي لو حَوّل المُبرمِجُ الكودَ التالي:

```
لو الإجراء(5  10) > 15 و س :
    \ كود /
```

إلي ما يَلِي:

```
رقم ص = الإجراء(5  10)
لو ص > 15 و س :
    \ كود /
```

فسيستفيد من التحسينات المُمكِنة.

سبب عدم صنع إجراءاتٍ حرةٍ قياسيةٍ يتم استدعاء كل واحدٍ منها عند: قراءة قيمة حاوية. أو تعديل قيمة حاويةٍ من الحاويات. كرد فعلٍ لما فعله المبرمج :

مثلاً يكون الإجرائين المُرتبطين بالمتغير (الرقم) كما يلي:

```
رقم الرقم
إجراء حر تعديل.الرقم :
    \ كود /
إجراء حر قراءة.الرقم :
    \ كود /
```

بحيث تُحاكِي هذه الإجراءات عمل المُغلِّفات properties.

الجواب:

- سيتم استدعاء إجراءات التعديل بعد التعديل الفعلي في الحاوية، و هذا أقل قوةً من الحالة التي يتم فيها فحص القيمة قبل إسنادها للحاوية (مثلما يحدث في حالة الأغلفة أو إجراءات get و set العادية التي سنُسمّيها الإجراءات المُغلّفة منذ الآن فصاعداً).

و لمُحاكاة الأسلوب الأخير سوف تتعقد الإجراءات الحرة، و تتعقد كيفية عملها، و كذلك كيفية إسناد القيم للحاويات نفسها بما يجعل تكلفة الأمر أكبر بكثيرٍ من فائدته التسهيلية !

- يُمكِن للإجراءات المُغلّفة القيام بهذا الدور بمُنتهَي الكفاءة بدون الحاجة لقواعدٍ إضافيةٍ في اللغة و المُترجِم و المُفسِر، و في نفس الوقت تحتفظ هذه الإجراءات بميزة أنه يُمكِن إعطاء كل واحدٍ فيها مُستوَي وصولٍ و/أو نوعٍ و/أو استنساخٍ مُختلِف عن الإجراء الآخر و/أو عن الحاويةِ التي يتعلق بها عمله.

- سيتَسبب وُجود هذه الإجراءات الحرة القِياسِية في عملِ عِبءٍ مُخِيف في زمن التنفيذ؛ نظراً لحاجتنا الماسَّة لتوفير القُدرة علي التعامل معها دوماً في زمن التنفيذ لكل الحاوِيات التي في المكتبات أو في برنامج المُستخدِم، و فكرة (فحص وجود التعامل مع هذه الإجراءات في كود البرنامج لتحديد: هل يوجد سببٌ لوضع كودٍ لها في زمن التنفيذ أم لا ؟) سوف يُؤدِّي تطبيقها إلي بطء عملية الترجمة بشدة، بمقدارٍ يتناسب مع حجم المكتبات المُستخدمة و عدد الحاويات فيها ! و هو الأمر الذي يستحيل قبوله عملياً.

- سوف تظهر أسئلةٌ مثل :
  - هل التعامل مع حاويةٍ داخل كائن صِنفٍ يُعتبر تعاملاً مع كائن الصنف، بحيث يُتطلّب معه إستدعاء رد الفعل الخاص بكائن الصِنف ؟
  - هل التعامل مع عنصرٍ في جدولٍ يُعتبَر تعاملاً مع مُؤشِّر الجدول، بحيث يُتطلّب إستدعاء رد الفعل الخاص بمُؤشِّر الجدول ؟

  و للإجابة عن هذه الأسئلة سوف نُضطر لوضع المزيد و المزيد من القواعد، مما يعني تعقيد نَحْو اللغة بشدة.

من محاسن وجود الوراثة المتعددة مع إفراد كل صنفٍ خارجيٍ في ملفٍ مُنفصل :

في حالة وُجود صِنْفٍ فيه عددٌ كبيرٌ من الإجراءات، و تنقسم هذه الإجراءات إلي مجموعاتٍ وظائفيةٍ مُختلِفة: فإنه يُمكِن وضع الإجراءات الخاصة بكل مجموعةٍ وظائفية في صِنْفٍ خاصٍ بها، ثم تتم وراثة كل تلك الأصناف في صِنْفٍ واحدٍ يُشكِّل هو البديل العملي للصنف القديم المُتخَم.

من فوائد جعل تعليقات إبداع علي الهيئة \ تعليق / :

- صِيغةٌ مُوحّدة يُمكِن استعمالها في حالة السطر الواحد أو الأسطر المتعددة، و بالتالي أسهل في الحفظ.
- يُمكِن لتعليقٍ احتواء أي عددٍ من التعليقات الأخري، حيث يتم قراءة عدد العلامات \ و انتظار عددٍ يُماثِلها من علامات / و هذا يُؤدّي إلي سهولة تحويل أي مِساحاتٍ من الأكواد إلي تعليقٍ دون التضارب الذي يحدث في حالة عائلة الـC مع علامات /* */

فوائد منع الأعراض الجانبية في إجراءات الملف الرئيس :

- القُدرة علي تنفيذ الإجراءات علي التوازي مع عدم وُجود ذاكرةٍ مُشترَكةٍ يُخشَي منها علي إطلاق الحرية غير المتناهية للإجراءات في التوازي.
- إعتماد الإجراءات علي مُعامِلاتِها فقط يُمكّننا من استخدام تحسينات وقت الترجمة، مثل:

```
رقم ص = 10
رقم س = مضروب(ص)

إجراء (رقم س) مضروب(رقم ص) :
    لو ص = 1 أو ص = 0:
        س = 1
    غيره :
        س = ص × مضروب(ص-1)
```

فبدلًا من أن يتم تنفيذ هذا الإجراء في زمن التنفيذ فيُمكِن للمُترجِم تنفيذه في زمن الترجمة توفيراً للوقت.

- المُساعَدة علي جعل الإجراءات كياناتٍ قائمةٍ بذاتها يُمكِن نقلها من مكانها بسهولة؛ حيث يُساعِد ذلك علي البرمجة و التصميم بما يُشبه الحالات التي لا نعرف فيها شكل البرنامج النهائي، و إنما نَعرف بعض الخصائص التي نُريد بناءها و بالتالي نتمكن من بِناء الإجراءات الخاصة بها، و عندما نُريد ترتيب الإجراءات علي نحوٍ مُختلِفٍ يُمكِننا ذلك بدون القلق من كَوْن الإجراء مُرتبطٌ بمكانه بسبب الأعراض الجانبية التي تنتج عنه أو عن أحد الإجراءات الأخري.

- تُتِيح لنا إمكانية استخدام الحساب المُصغَّر lazy computation متي أردنا ذلك و وجدناه أنفع للأداء، حيث أنه ما دام الإجراء ليس له أي تأثيرٍ إلا علي مُخرَجاته، و لا يُؤثِّر بالأعراض الجانبية علي مُكوّنٍ آخرٍ؛ فإن ذلك معناه أن عدم استدعائه لن يُشكّل مُشكِلةً للبرنامج.

و رغم أن الإجراء قد يكون له أعراضٌ جانبيةٌ حتي و لو كان لا يُؤثِّر إلا علي مُخرَجاته (مثل تغيير حقل بياناتٍ في كائن صِنْف، أو ا ستدعاء إجراءٍ ذي أعراضٍ جانبيةٍ) إلا أن ذلك سهل التعامل معه بالقاعدة التالية :

(لو كان الإجراء لا يُؤثِّر إلا علي مُخرَجاته، و كل مُدخَلاته من النوع المُمرَّر بالقيمة: فيُمكِننا ا ستخدام الحساب المُصغَّر معه لو رغبنا في ذلك).

القاموس

بعض الإصطلاحات العربية المُستَخدمة ليست هي الأشهر في ترجمة مُرادِفاتها التقنية الإنجليزية، و إنما استخدمتُها لأنها أدق و أوضح من حيث المعني، بينما الترجمة المشهورة حَرْفِيةٌ أكثر من اللازم.

الإصطلاح العربى	الإصطلاح الإنقليزى المُكافِئ
صنف	class
استنساخ صِنْف	Class Instantiating
لقب	enumeration
إنتاجية	productivity
تعليق	comment
إجراء داخل صِنْف	method
كود أصلى	native code
كود محكوم	managed code
برمجة كائنية	Object oriented programming
برمجة وظائفية	functional programming
لغة تفسيرية	interpreted language
لغة مُترجَمة	compiled language
مُؤشِّر	Pointer
مُترجِم	compiler
مُتغيِّر	variable
مُعالجة الأغلاط	exceptions handling
مُعامِل	operator
مُغلِّف	property
مُفسِّر	interpreter
مكتُوبية	writability
مُشيِّد	constructor
نُواة (في نظام التشغيل)	kernel
هياكِل	structs
واجِهة	interface
وِراثة	inheritance

delegate	وَسِيط
procedure	إجراء
Procedure call	استدعاء إجراء
Variable definition	إشْهار (تعريف) مُتغيّر
parameter	مُعامِل (دَخْل إجراء)
Reserved word	كلمة محجوزة
space	مسافة
indentation	إزاحة
namespcae	حَيّز أسماء
Multiple inheritance	وراثة مُتعددة
Code block	مساحة تكويدية = كتلة من الكود
Procedures overloading	تحميل الإجراءات
Exception throwing	إلقاء غلط
abstracting	التجريد
recompile	إعادة ترجمة
index	مُميّز
Side effects	الأعراض الجانبية
Structured query languages (SQL)	لغات الإستعلام المُهَيكَل
Digital image processing	مُعالَجة الصور الرقمية
header	تَرْوِيسَة
Track bar	عمود تَعَقُّب
browser	مُتصفِّح
portability	محمولية
GUI applications	برامج بواجهة رسومية
Console applications	برامج بواجهة نَصِّية
Clients applications	برامج عملاء
server	خادم
Servers applications	برامج خوادم
static typing	تنويع ثابت
Dynamic typing	تنويع مُتغيِّر

Safe programming	برمجة آمنة
unsafe programming	برمج غير آمنة
GUI designer	مُصمِّم واجهة رسومية
struct	هَيْكل
task	مهمة
indexer	مُفَهْرِس
union	دامِج
implementation	بِناء
Header file	ملف ترويسة
stack	مُكدِّس
Backword compatibility	توافق عكسي
Functions pointers	مؤشرات الدول
Compile time	زمن الترجمة
Run time	زمن التنفيذ
Generic codes	الاكواد العامة
series	مُتسَلْسِلَة
lazy computation	الحساب المُصغَّر

المصادر

- learning OpenCV (computer vision with OpenCV library). By: gary bradski & adrian kaebler. O'REILLY.
- Professional C Sharp 3^{rd} Edition – Wrox.Press -2004 by:
    a. Simon Robinson
    b. Christian Nagel
    c. Jay Glynn
    d. Morgan Skinner
    e. Karli Watson
    f. Bill Evjen
- C Sharp, from Wikipedia, the free encyclopedia.
- C# How To Program (Redistilled In One Book) by: Deitel.
- Slides named "Introduction to C#" Anders Hejlsberg, Distinguished Engineer Developer, Division Microsoft Corporation
- (الكامل فى السى شارب) لمُؤلِّفٍ حقيقةً لا أعرف اسمه لكنه يضع لنفسه الرمز OMS
- Introduction to C# The New Language for Microsoft .NET, by H.Mössenböck, University of Linz, Austria.
- Beginning Programming with Java for Dummies, 2^{nd} Ed – [Wiley], by: Barry burd.
- Introduction to Java programming: comprehensive version, by: Y. Daniel Liang.—6'Th Ed.
- Comparison of C Sharp and Java, From Wikipedia, the free encyclopedia.
- Java (programming language), From Wikipedia, the free encyclopedia.
- (برمجة الحاسب)، المملكة العربية السعودية، المُؤسسة العامة للتعليم الفنى و التدريب المِهنـى، الإدارة العام لتصميم و تطوير المناهج.

- C++ Data Structures 3rd.Edition, by: Nell dale, university of Texas, Austin.
- كتاب (Visual Basic للجميع) الإلكترونى للمؤلف: تركى العسيرى.
- كتاب (إبدأ مع لغة أوبجكت با سكال Object Pascal)، للأخ: معتز عبد العظيم الطاهر (أبو إياس).
- كتاب (الخطوة الثانية مع أوبجكت با سكال – صناعة البرمجيات)، للأخ: معتز عبد العظيم الطاهر (أبو إياس).
- كتاب (بايثون بلمسة)، للمؤلف: مصطفي فرحات.
- ترجمة كتاب (a byte of python) للمؤلف: Swaroop C H، ترجمة: أشرف علي خلف.
- F# Eye for the C# guy, by: Leon Bambrick, secretGeek.net
- Why F# is like it is, by: Robert Pickering, ALTI
- (البرمجة بلغة با سكال)، للمؤلف: وجدى عصام عبد الرحيم
- Eiffel Analysis, Design and Programming Language, Standard ECMA-367, 2nd Edition – June 2006
- unix haters handbook, Edited by Simson Garfinkel, Daniel Weise, and Steven Strassmann, Illustrations by John Klossne. Published by :IDG Books Worldwide, Inc.
- الورقة العلمية (جملة goto تعتبر ضارة goto statement cosidered harmeful)، للبروفيسور:edsger w.dijkstra.
- الورقة العلمية (ملاحظاتٍ حول تصميم لغة البرمجة hints on programming language design) للبروفيسور C. A. R. HOARE.
- مجلة (مجتمَع لِينُوكس العربى)، العدد 7 الصادر فى رمضان 1430هـ (الموافق أغسطس 2009م).
- The Measured Cost of Conservative Garbage Collection by B. Zorn

- كُتيّب (مقدمة إلي لغة ada)، إعداد: مهندسة تولاي شاهين، جامعة حلب، كلية الهندسة المعلوماتية.
- The Book (ADA PROGRAMMING), by Wikibooks contributors, Developed on Wikibooks the open-content textbooks collection.
- موقع: stackoverflow.com
- مواضيع مختلفة في Wikipedia, the free encyclopedia.

السِّيرة الذَّاتية



البَيانات الشخصية:

الإسم: م. وائل حسن محمدعلى عبدالمنعم (كنيتي: أبوإياس)

الجنسية: مصرى

محل الميلاد: أُسْوَان- كُوم اُمْبُو

الحالة الإجتماعية: أعزب

تاريخ الميلاد: 22/10/1987

الموقع الشخصي:

http://afkar-abo-eyas.blogspot.com

البريد الإليكتروني:

Wael_hasan_87@yahoo.com

المُؤهِّلات الدراسية و التخصص:

جامعة جنوب الوادى

كلية الهندسة و التكنولوجيا بأُسْوَان

قسم كهرباء

شعبة حاسبات و نظم

مشروع التخرج: game development using C# and XNA) .(library

تقدير المشروع : إمتياز التقدير العام: جيد

التخصص الحالي: مُهندِس برمجيات في مجال تصميم و بناء لغات البرمجة. (بناء لغات البرمجة يتضمن تصميم و بناء المُترجِمات compilers و المُفسِّرات interpreters و غيرهما من الأدوات الأخري).

و مُصمِّم لغة البرمجة العربية الإحترافية (إبداع) ، و باني المُفَسِّر القياسي (أُبْدِع obde3) لها.

المهارات الحاسوبية:

- لغات البرمجة:

•Java

•C#

•C++

•Python

•JavaScript

•Object Pascal

•Fortran

•matlab

•VB.NET

•ruby

- نظام التشغيل المُفضَّل هو القنو/لينوكس GNU/linux , و خاصةً توزيعة kubuntu.

- أقوم منذ شهر رجب لعام 1433هـ (الموافق لشهر يونيو لعام 2012م) بنشر مقالاتٍ تقنيةٍ علي موقع (وادي التقنية Itwadi.com).

خبرة العمل :

- نوفمبر، 2010 إلى سبتمبر، 2011
بَاحِثٌ مُساعِدٌ فى مركز أبحاث الذكاء الاصطناعى و الروبوت Center for Artificial Intelligence and RObotics - C.A.I.R.O بكُلِّيَّة العلوم بأُسْوَان.

- سبتمبر، 2011 إلي الأبد إن شاء الله عز و جل
مُتفرِّغٌ لمشروع (البرمجة بإبداع).

أدوات إنتاج هذا الكتاب

- توزيعة kubuntu من نظام التشغيل قِنُو/ لِينُوكس GNU/linux.

- برنامج LibreOffice writer لكتابة الوثائق.

- برنامج Gwenview لعرض الصور و المُعالَجة البسيطة لها.

- برنامج gimp لتعديل الصور باحتراف.

- برنامج kolourpaint لتعديل الصور بشكل بسيط.



رِسَالَةُ البَرْمَجَةِ بِإِبْدَاع

(رسالة البرمجة بإبداع ™) هي رحلةٌ علميةٌ مُشوّقة يتم فيها التحدث باستفاضةٍ عن:

مشروع البرمجة بإبداع ™ و الدوافع وراءه و أهدافه المُتنوعة.

شرح لغة البرمجة العربية الإحترافية إبداع ™ علي نحوٍ مُختصَر.

ثم شرح الأسباب المنطقية وراء ذلك التصميم بالتفصيل في عشرات الصفحات التي أظن أنها ستكون مُمتِعة لمن يُحب البرمجة و الإبتكارات الجديدة فيها ، لأنها تطوف في العوالم البرمجية المُختلفة و تُرِيكم ما وراء الكواليس قدر الإمكان حتي يتبين وجه التعارض أو التوافق معه.

م. وائل حسن محمدعلي (أبو إياس)